Spis treści

1. Wstęp

a. Cel projektu

Projekt zakłada stworzenie aplikacji mobilnej tj. przeznaczonej na urządzenia mobilne oraz przygotowanie dokumentacji przedstawiającej jej założenia, proces powstawania, jak również omówienie finalnego produktu. Zadaniem aplikacji jest ułatwienie jej uzytkownikowi nauki gry na gitarze poprzez:

- umożliwienie samodzielnej nauki gry bez konieczności wcześniejszego poznania nut bądź opanowania tabulatury.
- udostępnienie przygotowanych utworów do nauki, co niewątpliwie okaże się dla użytkownika bardziej atrakcyjne niż nauka podstawowych ćwiczeń.
- obszerną bazę chwytów, mogącą służyć użytkownikowi nawet na średnim poziomie zaawansowania nauki gry na gitarze.
- wykorzystanie aspektu mobilności, tzn. umożliwienie pełnego korzystania z aplikacji w każdym miejscu, także bez dostępu do internetu.

b. Najpopularniejsze sposoby nauki gry na gitarze

Nauka gry na gitarze jest procesem czasochłonnym oraz wymagającym determinacji. Cieszy się ona jednak na tyle dużym powodzeniem, że obecnie możemy wyróżnić kilka popularnych sposobów mających za zadanie ją ułatwić oraz uprzyjemnić:

- Szkoła muzyczna

Pomoc w nauce gry na gitarze jest oferowana w szkołach muzycznych, które strukturą przypominają np. państwowe szkoły podstawowe. Składa się ona z dwóch stopni – w sumie trwających 10 lat. Nauka obejmuje kompleksowe omówienie oraz praktykę wszystkich elementów związanych z grą na gitarze.

Przedmioty nauczane w szkole muzycznej zakresem wykraczają jednak poza informacje oraz umiejętności bezpośrednio z grą związne. Przykładami takich przedmiotów są historia muzyki, kształcenie słuchu czy rytmika. Mimo braku ścisłego powiązania z grą na gitarze spełniają one kluczową rolę w perspetywie wieloletniej nauki. Wiedza oraz umiejętności

wchodzące w ich zakres są niezbędne do zrozumienia i opanowania utworów zaawansowanych technicznie.

Nauka samej gry na gitarze odbywa się z wykorzystaniem nut. Kompozycje wchodzące w program nauczania pochodzą z różnych epok oraz są dobrane tak, by kształcić wszystkie elementy gry na gitarze. [11]

Podsumowując, nauka w szkole muzycznej oferuje kompleksowe kształcenie gry na gitarze, jak również dostarcza wiedzę teoretyczną, która przy skomplikowanych kompozycjach jest konieczna. Szkoła wymaga jednak poświęcenia dużej ilości czasu na tematy, które dla uczniów mogą okazać się nie interesujące. Obowiązkowa jest również opłata czesnego.

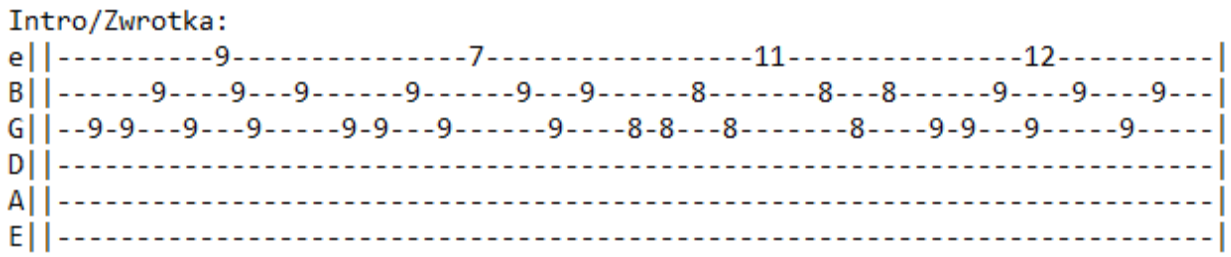
- Ognisko muzyczne

Kolejnym sposobem mającym na celu ułatwienie nauki gry na gitarze jest uczęszczanie do ogniska muzycznego. W odróżnieniu od szkoły muzycznej tematyka nauk jest ograniczona i często może być ona uzgadniana z nauczycielem. Zajęcia odbywają się zazwyczaj w formie indywidualnej. Każde spotkanie jest płatne. [12], [13], [14]

- Nauka samodzielna z wykorzystaniem tabulatury

Jeżeli początkujący gitarzysta czuje się na siłach by uczyć się samodzielnie, jednak nie chce korzystać z nut, może spróbować opanować czytanie tabulatury [Rys. 1], co powinno okazać się prostsze. Dzięki internetowi każdy ma dostęp ogromnej ilości utworów zapisanych z jej wykorzystaniem.

Samodzielna nauka wymaga ponadprzeciętnego słuchu muzycznego, ponieważ nie możemy skorzystać z pomocy nauczyciela, który taka zdolność powinien posiadać.



Rys. 1 Fragment gitarowej tabulatury

c. Przykładowe aplikacje mobilne do nauki gry na gitarze

Wraz ze wzrostem popularności urządzeń mobilnych typu smartfon czy tablet powstawać zaczęły aplikacje mające na celu ułatwienie nauki gry na gitarze na nie dedykowane. Dzięki rozwojowi technologicznemu możliwe stało się stworzenie zaawansowanych programów, które poziomem skomplikowania nie odstępują programom napisanym na komputery stacjonarne.

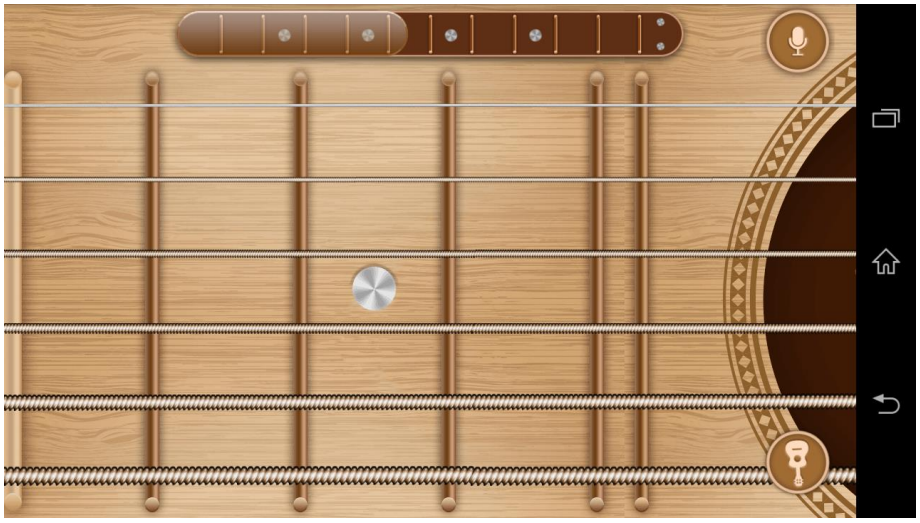
W tym momencie w usłudze *Google Play* dostępnych jest około 200 aplikacji do nauki gry na gitarze. Liczby pobrań poszczególnych aplikacji sięgają nawet 10 milionów. Świadczy to o dużej ilości osób, które chcą uczyć się gry na gitarze z wykorzystaniem urządzeń mobilnych.

Poniżej w skrócie przedstawiono kilka przykładowych aplikacji w celu zobrazowania, jak różni deweloperzy wychodzą na przeciw tym oczekiwaniom:

- *Gitara* od *Fotoable, Inc.*

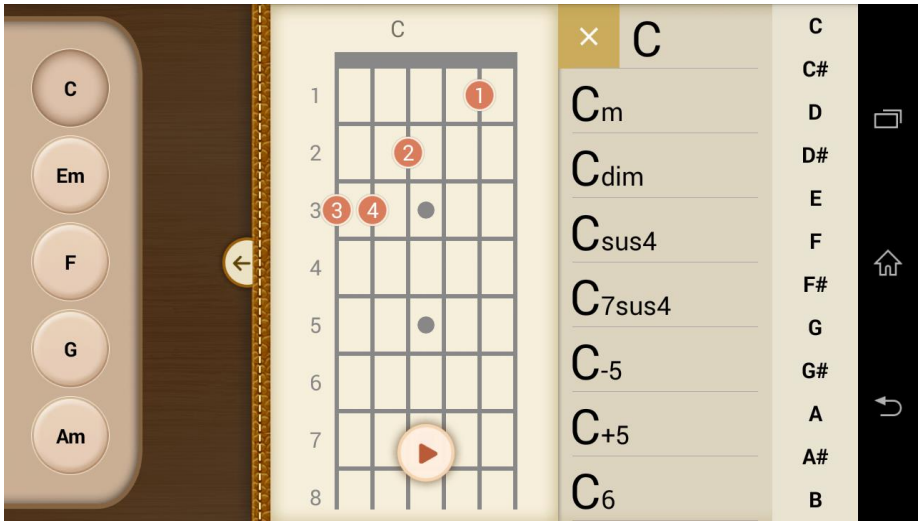
Aplikacja umożliwia korzystanie z całego gryfu[1] gitary. Ponieważ nie jest on w stanie zmieścić się na ekranie urządzenia w pełnej długości, w górnej części znajduje się pasek dzięki któremu możemy wybrać interesującą nas jego część [Rys. 2]. Obsługiwana jest gra wielu nut po sobie poprzez przeciągnięcie palca po ekranie. Istnieje również możliwość nagrania swojej gry do plików MIDI. Aplikacja nie posiada jednak funkcji ich prezentacji na gryfie.

[1] „gryf – drewniany element występujący w większości instrumentów strunowych (m. in. skrzypcach, w gitarze) nad którym rozpięte są struny. Pod strunami znajdują się progi wykorzystywane do wydobywania konkretnych dźwięków.” [15]



Rys. 2 Tryb gry dowolnej

Drugim z oferowanych trybów nauki jest baza chwytów. Zawiera ona prawie 500 akordów[2], co jest imponującą liczbą, nawet jeśli wziąć pod uwagę, że większość z nich jest bardzo rzadko wykorzystywana w zdecydowanej większości utworów. Z prawej strony ekranu możemy wybrać tonację[3] oraz interesujący nas chwyt [Rys. 3]. W centrum widzimy w jaki sposób dany akord zagrać, oraz naciskając strzałkę u dołu, usłyszymy jego brzmieniei.



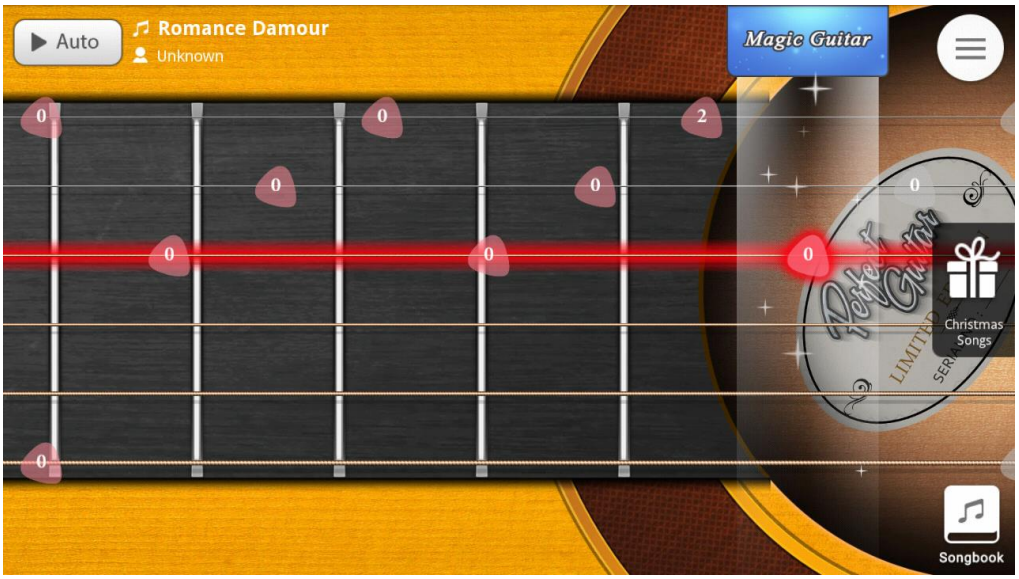
Rys. 3 Tryb nauki akordów

[2] „akord (chwyt) – współbrzmienie przynajmniej trzech dźwięków.” [7]

[3] „tonacja – przynależność materiału dźwiękowego do konkretnej gamy durowej lub molowej, na której jest on oparty.” [16]

- *Gitara + (Guitar)* od *rubycell*

Aplikacja posiada kilka funkcji, jednak tylko jedna pomaga w nauce gry na gitarze. W prawej części ekranu umieszczony jest obszar *"Magic Guitar"*, na który z przeciwnej strony przesuwają się nuty. Gdy znajdą się one w białym pasku należy nacisnąć odpowiadające im pole [Rys. 4]. Należy zwrócić uwagę, iż ten sposób nauki nie ma całkowitego odzwierciedlenia w grze na prawdziwej gitarze. Nuty naciskamy zawsze w kolumnie *"Magic Guitar"*, nie na progach. Istnieje jednak możliwość automatycznego odtwarzania utworu oraz regulacji tempa, a nuty zawierają cyfrę odpowiadającą numerowi progu na którym się znajdują. Wykorzystując więc prawdziwą gitarę można w ten sposób próbować uczyć się gry ponad 100 utworów dostępnych w aplikacji.



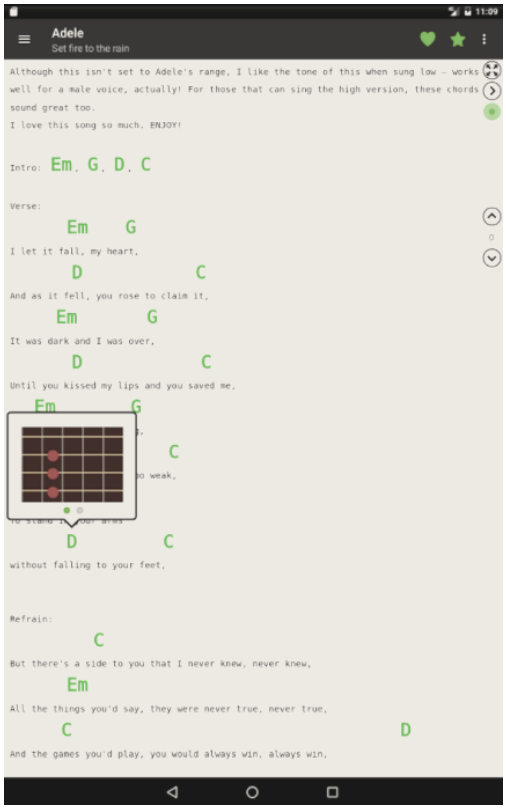
Rys. 4 Tryb nauki utworów

- *Guitar chords and tabs* od *Xssemble*

Aplikacja posiada dostęp do około 800,000 piosenek, które prezentuje w formie tekstu oraz akordów nad odpowiednimi słowami, informując w ten sposób kiedy należy je zagrać. Konieczna jest jednak uprzednia znajomość piosenki, bądź gra podczas gdy jest ona odtwarzana w tle z innego urządzenia, by wiedzieć jakie jest tempo utworu oraz jakie bicie

gitarowe[4] zastosować. Naciskając nazwę chwytu uzyskamy informacje o nutach do niego należących [Rys. 5].

Można zauważyć, że ten tryb nauki jest jedynie udogodnioną wersją bazy akordów z innych aplikacji, przez co umożliwia naukę jedynie gry akompaniamentu.



Rys. 5 Wybrany chwyt *D*

Podsumowując, wszystkie omówione aplikacje umożliwiają darmową pomoc w samodzielnej nauce gry na gitarze. Zakres wiadomości i umiejętności możliwych do zdobycia, oraz sposób nauczania są jednak znacznie uboższe w porównaniu do kształcenia w szkole muzycznej czy nawet w ognisku muzycznym.

Tryb gry dowolnej można potraktować jako funkcję, która w uproszczony sposób jedynie zastępuje gre na prawdziwej gitarze. Ma ona jednak minimalny potencjał w kontekście jej nauki.

[4] bicie gitarowe – uderzanie strun podczas gry akompaniamentu.

Baza akordów, szczególnie w formie zaimplementowanej w ostatniej omawianej aplikacji, jest funkcją przydatną gdy zależy nam na nauce gry akompaniamentu do piosenek. Nie jest to jednak metoda, która obejmuje naukę utworów pisanych na gitarę.

W części umożliwia ją nauka z aplikacją *Gitara+*, ale ponieważ progi gitary reprezentowane są wyłącznie w postaci cyfr umieszczonych na przesuwających się nutach, rzeczywiste odzwierciedlenie na prawdziwej gitarze ma jedynie umieszczenie nut na poszczególnych strunach.

2. Specyfikacja wymagań

Aplikacja stworzona na potrzeby tego projektu ma za zadanie oferować umożliwienie nauki utworów z uwzględnieniem pustych strun oraz pierwszych czterech progów gitary. Ograniczenie ich ilości wynika z chęci zlikwidowania konieczności przesuwania ekranu, tak jak było to zaimplementowane w jednej z przykładowych aplikacji w poprzednim punkcie. Nie ogranicza to jednak zbyt funkcjonalności aplikacji, szczególnie w przypadku osób dopiero zaczynających naukę.

a. Docelowy użytkownik

Docelowym użytkownikiem jest przede wszystkim osoba, która decyduje się na samodzielną naukę gry na gitarze, bez uczestnictwa w szkole muczycznej, bądź korzystania z nauk prywatnego nauczyciela. Użytkownik powinien wiedzieć na czym polega gra na gitarze, oraz umieć posługiwać się urządzeniami mobilnymi z systemem *Android*. Dzięki części utworów, przygotowanych dla osób już potrafiących grać na gitarze, oraz możliwości importowania własnych plików *MIDI*, aplikacja może służyć nawet zaawansowanym gitarzystom. Baza akordów zawiera zarówno chwyty proste, łatwo rozpoznawalne na przykład z piosenek granych na co dzień w radiu, jak również akordy trudniejsze, których opanowanie i zastosowanie okaże się przydatne dopiero po skończeniu początkowego etapu nauki.

b. Wymagania funkcjonalne

- nauka utworów – użytkownik ma możliwość wyboru z pośród 14 przygotowanych utworów do nauki, które poziomem trudności obejmują kompozycje dla początkujących oraz średnio zaawansowanych gitarzystów. Istnieje także możliwość importowania własnych piosenek w formacie *.mid*. Wybrany utwór jest odtwarzany, a wszystkie nuty pojawiają się w odpowiednich polach na gryfie. Użytkownik może dostosować tempo odtwarzania piosenki oraz włączyć metronom, który będzie je wystukiwał. Jeśli utwór zawiera fragmeny, które należy zagrać biciem gitarowym, aplikacja wyświetla strzałkę informującą o kierunku.
- nauka akordów - dla każdej tonacji dostępne do nauki są chwyty, których możliwe jest zagranie wykorzystując pierwsze cztery progi gitary oraz puste struny. Po wybraniu interestującego nas akordu zostanie on zaprezentowany na gryfie gitary, oraz odtworzony.
- gra dowolna – umożliwia grę w aplikacji poprzez naciskanie odpowiednich pól na gryfie. Jest także częścią poprzednich dwóch funkcji.
- funkcja zmiany dźwięku gitary – pozwala zmienić dźwięk nut granych przez użytkownika odpowiadający strunom nylonowym bądź stalowym.

c. Wymagania niefunkcjonalne

- możliwość zainstalowania oraz poprawnego działania aplikacji na większości urządzeń z systemem Android, poprzez odpowiednią implementację funkcjonalności oraz zastosowanie bilbliotek kompatybilnych ze starszymi wersjami systemu.
- intuicyjny oraz prosty w obsłudze graficzny interfejs użytkownika.
- odporność na błędy mogące wynikać z importu niekompatybilnych bądź uszkodzonych plików MIDI.
- odporność na błędy spowodowane wolnym działaniem urządzenia co może mieć wpływ np. na zsynchronizowane odtwarzanie utworu oraz metronomu.
- krótki czas uruchamiania aplikacji.
- czytelny kod aplikacji napisany w logicznej strukturze, co umożliwi jej dalszy, łatwy potencjalny rozwój.
- teksty w aplikacji przygotowane w języku polskim oraz angielskim, wyświatlane w aktualnym języku systemowym.

3. Narzędzia pracy oraz technologie

Aplikacja została napisana z wykorzystaniem komputera stacjonarnego z procesorem od firmy *AMD*, na którym niemożliwe jest uruchomienie funkcji *SVM (ang. Secure Virtual Machine).* Funkcja ta zapewnia zestaw rozszerzeń sprzętowych, dzięki którym możliwe jest skuteczne oraz sprawne uruchamianie maszyn wirtualnych. Z tego względu zdecydowano się na pisanie aplikacji testując ją z wykorzystaniem prawdziwego urządzenia z systemem *Android*, smartfonu *Sony Xperia Z1 Compact*, zaprezentowanego w 2013 roku.

Zintegrowane środowisko programistyczne wykorzystane do napisania kodu aplikacji to *Android Studio*[17] od *Google* i *Jetbrains*. Dzięki funkcjom pozwalającym na planowanie intefejsu aplikacji na bardzo wczesnym etapie rozwoju aplikacji, jak również dostarczenie możliwości zastosowania gotowych wzorców klas proces tworzenia programu przebiegł sprawnie pomimo braku wcześniejszego doświadczenia w programowaniu na system Android.

Zdecydowana większość elementów graficznych aplikacji została utworzona z użyciem programu *GIMP (ang. GNU Image Manipulation Program)*[18]. Jest to prosty w obsłudze editor grafiki rastrowej, którego zaawansowane możliwości pozwoliły jednak na stworzenie graficznego interfejsu użytkownika tak, by umożliwiał zaimplementowanie wszystkich zamierzonych funkcjonalności oraz spełniał wszystkie wymagania niefunkcjonalne.

Aplikacja przeznaczona jest na urządzenia z systemem *Android*. Głównym czynnikiem, który zdecydował o wyborze tego systemu a nie *Apple iOS* wykorzystywanego przez urządzenia od firmy *Apple* była możliwość pisania aplikacji z użyciem jezyka *Java*. Pozwala on na intuicyjne programowanie obiektowe, które w sytuacji poznawania nowego środowiska programowania, umożliwiło łatwiejszy start w pracy nad projektem.

a) Plik dźwiękowy MIDI

Język *Java* posiada wbudowany pakiet do obsługi plików MIDI, które w tej aplikacji są źródłem wszystkich dźwięków.

MIDI jest formatem dźwiękowym zawierającym specyficzny zapis nutowy [Rys. 6] danego utworu, nie spróbkowany dźwięk, jak np. format *mp3*. Jego budowa wygląda w następujący sposób:

- Plik może składać się z jednej bądź większej ilości tzw. **ścieżek** (ang. tracks).
- W ścieżce umieszczane są tzw. **wydarzenia** (ang. events). Każde z nich posiada parametr określający moment jego aktywacji wyrażony w tzw. **cykach** (ang. ticks).
- Dla każdej ścieżki ustalony jest wspólny parametr **PPQ** (ang. Pulses Per Quarter), wyznaczjacy długość ćwierćnuty w cykach.

Do najważniejszych wydarzeń należą:

- *NoteOn* oraz *NoteOff* – odpowiedzialne za rozpoczęcie oraz zakończenie gry danej nuty. Posiadają parametry takie jak:
  - *On/Off Tick* – określa w cykach kiedy dana nuta ma się rozpocząć/zakończyć.
  - *Duration* – czas trwania nuty wyrażony w cykach. Nie jest on konieczny, ponieważ zakończenie nuty jest aktywowane wydarzeniem NotOff.
  - *Note* – przyjmuje wartości od 0 do 127 i określa wysokość danej nuty[5]. Przykładowo parametr *Note* ustalony na 50 oznacza nutę D3 tj. dźwięk D w trzeciej oktawie.
  - *Velocity* – przyjmuje wartości od 0 do 127 i określa głośność danej nuty.
- *TempoChange* – wydarzenie mające na celu ustalenie tempa, jakie będzie obowiązywać od danego cyku wyrażone w *BPM (ang. Beats Per Minute). BPM* można rozumieć jako liczbę ćwierćnut przypadających na jedną minutę.
- *TimeSignature* – wydarzenie odpowiedzialne za ustalenie metrum[6] utworu. Wartość pola określającego moment aktywacji wydarzenia powinna być równa wartości początku lub końca taktu[7], ponieważ zmiana metrum możliwa jest tylko w tych miejscach.
- *ProgramChange* – pozwala wybrać na jakim instrumencie każda nuta od wyznaczonego cyku zostanie odtworzona.

[5] „wysokość dźwięku (nuty) – jedna z podstawowych cech dźwięku, której wartość możemy wyrazić w *Hz*; pozwala usystematyzować poszczególne dźwięki od „niskich" po „wysokie"." [7]

[6] „metrum - określenie, ile i jakich wartości znajduje się w każdym takcie utworu lub jego części, np. ¾ oznacza, że w takcie znajdują się 3 ćwierćnuty." [7]

[7] „takt – schemat metryczny, który porządkuje całość lub fragment utworu, wyznacza, ile i jakich wartości rytmicznych znajduje się w schemacie (takcie)." [7]

```
 00000 Time Signature = 4/4
 00000 Beats/Minute = 135
 00000 Channel 1 Piano
 00000 Channel 2 Piano
+00000 Piano[2] D# 39 (90) 40
+00000 Piano[2] D# 51 (90) 40
+00000 Piano[1] D# 63 (85) 07
-00007 Piano[1] D# 63
+00007 Piano[1] E 64 (85) 07
-00014 Piano[1] E 64
+00014 Piano[1] F 65 (85) 07
-00021 Piano[1] F 65
+00021 Piano[1] F# 66 (85) 07
-00028 Piano[1] F# 66
+00028 Piano[1] G 67 (85) 07
-00035 Piano[1] G 67
+00035 Piano[1] G# 68 (85) 06
-00040 Piano[2] D# 39
-00040 Piano[2] D# 51
 00040 Key Signature = C
+00040 Piano[2] A# 46 (85) 27
+00040 Piano[2] A# 34 (85) 27
-00041 Piano[1] G# 68
+00041 Piano[1] A 69 (90) 07
-00048 Piano[1] A 69
+00048 Piano[1] A# 70 (85) 07
-00055 Piano[1] A# 70
```

Rys. 6 Fragment MIDI z wydarzeniami ukazanymi w formie listy

Ponieważ plik MIDI sam w sobie nie zawiera żadnego dźwięku, wykorzystuje on specjalne biblioteki z nich się składające, dostępne na dysku danego urządzenia. Przykładowo, jeśli w pliku zostanie aktywowane wydarzenie *NoteOn*, a wartość aktualnego *ProgramChange* wynosi 25, odpowiadająca gitarze ze strunami nylonowymi, komputer wyszuka w bibliotece odpowiedni spróbkowany dźwięk i następnie przekaże go do syntezatora, który zajmie się odtworzeniem dźwięku.

a) Użyte biblioteki

API do obsługi *MIDI* zostało wprowadzone dla systemów *Android* dopiero w wersji 6.0. Według badań przeprowadzonych przez *popsci.com* we wrześniu 2017 roku liczba urządzeń mobilnych z tym lub nowszym systemem wynosi około 45%[19]. Oznacza to, że 55% użytkowników nie mogłoby skorzystać z aplikacji wykorzystujących *MIDI API* nawet jeśli ich smartfon lub tablet spełniałby wszystkie inne sprzętowe wymagania. Z tego powodu

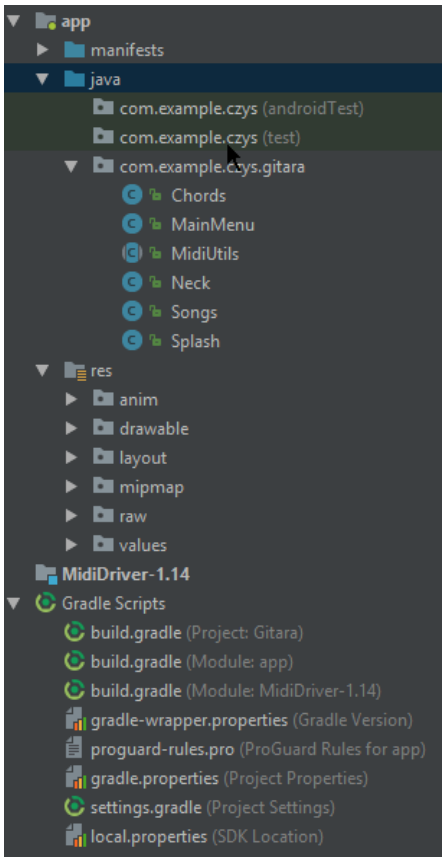
zdecydowałem się na skorzystanie z biblioteki *MidiDriver*[20], która pozwala korzystać z wszystkich istotnych funkcji dostępnych w pakiecie *javax.sound.midi* na komputerach stacjonarnych. Została ona zbudowana na bazie innej biblioteki, *Sonivox EAS*, i wykorzystuje spróbkowane dźwięki oraz syntezator w niej zawarte. Z użyciem *MidiDriver* możliwa jest na przykład konstrukcja wydarzeń *NoteOn* i *NoteOff* w czasie rzeczywistym.

Drugą zewnętrzną biblioteką użytą w aplikacji jest *Android MIDI Library*[21]. Udostępnia ona szereg klas oraz funkcji do pracy z plikami *MIDI*. Możliwe dzięki niej jest ich interpretacja, edycja czy ładowanie z oraz zapis do karty pamięci.

4. Przedstawienie budowy aplikacji

a. Struktura katalogów i plików

Struktura katalogów oraz plików projektu przedstawia się w następujący sposób [Rys. 7]:



Rys. 7 Widok na strukturę projektu z programu *Android Studio*

- *manifests* – folder ten zawiera jeden plik *AndroidManifest.xml* z podstawowymi informacjami o aplikacji, takimi jak wykorzystywane pozwolenia, nazwa, ikona czy startowa *Aktywność*.
- *java/com.example.czys.gitara* – tu znajdują się wszystkie klasy *Java* tworzące aplikację spełniające funkcje:
    - *Chords* – nauki akordów.
    - *MainMenu* – głównego menu aplikacji widocznego po ekranie startowym.
    - *Neck* – gry dowolnej.
    - *Songs* – nauki utworów.
    - *Splash* – wyświetlania ekranu startowefo przy uruchomienu aplikacji.
    - *MidiUtils* – klasy pomocniczej z metodami do obsługi plików MIDI. Jest ona wykorzystywana przez klasę *Songs.*
- *res* – zawiera podfoldery z materiałami wykorzystywanymi przez aplikację:
    - *anim* – znajdują się tu pliki *.xml* odpowiedzialne za generację animacji dla niektórych elementów interfejsu.
    - *drawable* – folder zawiera pliki graficzne w formatach *.png* i *.jpg* wykorzystywane w aplikacji, oraz pliki *.xml* do tworzenia elementów graficznych w procesie budowania projektu.
    - *layout* – zawiera pliki *.xml* odpowiedzialne za utworzenie szablonu graficznego dla każdej *Aktywności*. W nich zdefiniowany jest każdy element interfejsu tworzony na starcie danej *Aktywności*.
    - *mipmap* – tutaj znajdują się pliki graficzne wykorzystywane jako ikony aplikacji.
    - *raw* – zawiera czternaście plików *MIDI*, które podczas instalacji umieszczane są w katalagu *DCIM/Synthesian/* w pamięci wewnętrznej urządzenia.
    - *values* – folder ten zawiera pliki *.xml* przechowujące wartości często wykorzystywane przez aplikacje, bądź wartości umieszczone tu w celu ułatwienia procesu tworzenia aplikacji:
        - *colors.xml* – posiada wartości heksadecymalne niektórych używanych kolorów.
        - *strings-pl.xml* – zawiera tekst używany przez aplikację gdy wybrany język systemowy to język polski.
        - *strings-en.xml* – zawiera tekst w języku angielskim używany przez aplikację gdy wybrany język systemowy nie jest polskim.
        - *style.xml* – podstawowy plik *.xml* nie posiadający żadnej funkcji. Został utworzony jedynie ponieważ *AndroidManifest.xml* musi posiadać zdefiniowany parametr *theme*, z odwołaniem do pliku gdzie motyw ten jest utworzony.

- *MidiDriver-1.14* – tutaj znajdują się spróbkowane dźwięki wykorzystywane przez sterownik zawarty w bibliotece.
- *Gradle Scripts* – zawiera skrypty *Gradle* odpowiedzialne za automatyzację budowania aplikacji, od procesu kompilacji aż do instalacji na wybranym urządzeniu testującym bądź wirtualnej maszynie.

b. Implementacja aplikacji

Aplikacja została zaimplementowana zaczynając od funkcji gry dowolnej. Następnie utworzone zostało menu główne aplikacji oraz funkcje nauki gry akordów i utworów. Na samym końcu zaimplementowane zostały możliwości zmiany dźwięku gitary oraz ekran startowy. Poniżej znajduje się szczegółowy opis każdej z klas aplikacji:

- *Splash* – klasa odpowiedzialna za wyświetlenie ekranu startowego przy uruchamianiu aplikacji. Została utworzona, ponieważ ładowanie głównego menu jako pierwszej *Aktywności* prowadziło do wyświetlania białego ekranu przez około 3 sekundy, zanim menu zostało utworzone. Zawiera jedno pole SPLASH_DISPLAY_LENGTH określające jak długo ekran startowy ma być pokazywany, oraz dwie metody:
    - *onCreate(Bundle savedInstanceState)* – jak wszystkie metody o tej nazwie jest uruchamiana jednokrotnie, jako pierwsza po załadowaniu *Aktywności*. Przypisywany jest w niej odpowiedni szablon graficzny, określający wygląd ekranu startowego, oraz tworzony jest *Handler* obsługujący zakończenie działania tej *Aktywności* i rozpoczęcie ładowania *Aktywności MainMenu*. Przejście do niej następuje jednak dopiero po upływie ustalonego czasu oczekiwania.
    - *onWindowFocusChanged(boolean hasFocus)* – tak jak inne metody o tej nazwie jest uruchamiana gdy okno aplikacji stanie się aktywne np. poprzez utworzenie danej *Aktywności*. Ustalane w niej są flagi odpowiedzialne za przejście aplikacji w tryb pełnoekranowy imersywny, co oznacza, że pasek nawigacji oraz statusu są domyślnie ukryte. Aby je zobaczyć należy przesunąć palcem po ekranie od górnej lub dolnej krawędzi ekranu w kierunku jego środka. Tryb ten jest domyślnym trybem dla każdej z pozostałych *Aktywności*.

- *MainMenu* – klasa pełni rolę głównego menu aplikacji. Z niej możemy uruchomić inne Aktywności rezlizujące funkcje aplikacji oraz zmienić dźwięk gitary. Klasa zawiera następujące pola:

- *Intent startChords, startNeck, startSongs* – zmienne wykorzystywane przez niektóre metody do uruchamiania innych *Aktywności*.
- *Button []mainMenuButtons* – tablica, w której znajdować będą się widoki odpowiadające przyciskom głównego menu.
- *int instrument* – zmienna, która będzie przekazywana jako parametr do innych *Aktywności* z wartością określającą jaki dźwięk gitary został wybrany.

W klasie tej znajdują się następujące metody:

- *onCreate(Bundle savedInstanceState)* – metoda spełnia następujące funkcje:
  - Przypisywany jest w niej szablon graficzny.
  - Ustawiane są odpowiednie flagi, by aplikacja działała w trybie pełnoekranowym imersywnym. Pomiędzy przejściem z *Aktywności Splash* do *MainMenu* występuje krótka przerwa, podczas której żadna z nich nie jest aktywna, co skutkuje pojawieniem się pasków nawigacji oraz statusu. Dzięki flagom ustawionym już w metodzie *onCreate Aktywność* działa w trybie pełnoekranowym imersywnym nawet gdy nie jest jeszcze aktywna.
  - Następuje inicjalizacja utworzonych wcześniej zmiennych.

- *onWindowFocusChanged(boolean hasFocus)* – pełni taką samą rolę jak w klasie *Splash*.
- *onResume()* – gdy działanie *Aktywności* zostanie wznowione tło przycisków głównego menu zmieniane jest na domyślne.
- *onPause()* – metoda pauzuję główne menu gdy uruchamiana jest inna Aktywność.
- *onClick(View v)* – metoda ta powiązana jest w szablonie graficznym z przyciskami głónego menu. Naciśnięcie dowolnego z nich powoduje jej uruchomienie. Zawiera instrukcję wyboru *switch*, która zależnie od *ID* przycisku uruchamia odpowiedni *case*. *Case* dla przycisków związanych z innymi *Aktywnościami* powoduje zmianę tła danego przycisku oraz ich uruchomienie, z przekazaną ustaloną wartością parametru *instrument.*

- *Neck* – klasa odpowiada za pełnienie funkcji gry dowolnej. Wykorzystywana w niej jest biblioteka *MidiDriver*. Klasa posiada następujące pola:

- *MidiDriver midiDriver* – zmienna będąca sterownikiem tej biblioteki. Jest ona wykorzystywana do odtwarzania dźwięków pochodzących od kliknięć przycisków z gryfu gitary.
- *byte[] event* – tablica wykorzystywana do tworzenia wydarzeń, które są następnie obsługiwane przez sterownik *midiDriver*.
- *byte[][] notes* – tablica dwuwymiarowa zawierająca wartości nut w standardzie *MIDI* odpowiadające wszystkim nutom możliwym do zagrania z wykorzystaniem pierwszych czterech progów gitary oraz strun pustych.
- *ImageButton[][] notesButtons* – po skojarzeniu z odpowiednimi widokami szblonu graficznego zawiera pola służące do rozpoznawania jaki obszar ekranu został naciśnięty, w celu odtworzenia właściwej nuty.
- *TextView[][] notesButtonsClick* – tablica, która po skojarzeniu z odpowiednimi widokami służy wyświetlaniu animacji dla każdej odtwarzanej nuty.
- *boolean isNotePlaying* – tablica dwuwymiarowa zawierająca informacje o tym, czy dana nuta jest aktualnie odtwarzana.

Klasa zawiera następujące metody:

- *selectInstrument(int instrument)* – metoda tworzy wydarzenie *ProgramChange* w celu zmiany instrumentu na wartość pobraną z *Aktywności MainMenu*. Tablica *event* wypełniana jest odpowiednimi wartościami *byte*, by następnie zostać przekazana jako argument metody *write* wywołanej na sterowniku *midiDriver*.
- *onCreate(Bundle saved InstanceState)* – spełnia następujące funkcje:
  - Przypisuje *Aktywności* właściwy szablon graficzny.
  - Ustala flagę odpowiedzialną za dezaktywację automatycznego wygaszania ekranu.
  - Inicjalizuje wcześniej utworzone zmienne oraz tablice zmiennych kojarząc je z odpowiednimi elementami szablonu.
  - Dla pól pustych oraz przycisku powrotu aktywuje animacje powodujące ich pojawienie się na ekranie.
  - Elementom interfejsu przeznaczonym do interakcji z użytkownikiem przypisuje *onTouchListener*, umożliwiający zbieranie informacji o kliknięciach w metodzie *onTouch*.
  - Inicjalizuje zmienną *midiDriver* oraz wywołuje na niej metodę *setOnMidiStartListener*, co powoduje, że od tego momentu sterownik będzie obsługiwał wszstkie przekazane mu wydarzenia *MIDI*.
  - wywołuje metodę *selectInstrument*.

- *onResume()* – pełni taką samą funkcję jak w klasie *MainMenu*. Dodatkowo uruchamia także wstrzymany wcześniej *midiDriver*.
- *onPause()* – zatrzymuje *Aktywność* oraz sterownik *midiDriver*.
- *sendPlayNote(byte note)* – metoda tworzy wydarzenie *NoteOn*, w którym głośności przypisana jest maksymalna możliwa wartość, a wysokość ustalana jest z wykorzystaniem parametru *note* określanego przy wywołaniu metody.
- *sendStopNote(byte note)* – metoda tworzy wydarznie *NoteOff*. Zasada jej działania jest identyczna do metody *sendNoteOn*, lecz głośność otrzymuje wartość równą 0.
- *playNote(TextView notesButtonsClick, byte notes, int string, int notePosition)* – metoda odpowiada za zagrania nuty oraz obsługę czynności związanych z tym wydarzeniem:
    - Dla danej nuty uruchamiana jest animacja jej aktywności polegająca na zmianie przezroczystości właściwego elementu *notesButtonsClick*. Dzięki użyciu *ObjectAnimator* ostateczna wartość przezroczystości zostaje utrzymana aż do momentu wywołania animacji zmieniającej ją do stanu początkowego. W tym przypadku oznacza to, że element *notesButtonsClick* przestaje być widoczny gdy użytkownik podniesie palec z danej nuty.
    - Wywoływana jest metoda *sendPlayNote* z parametrem odpowiadającym wysokości dźwięku.
    - Wartość *isNotePlaying* dla danej nuty zmieniana jest na *true*.
- *stopNote(TextView notesButtonsClick, byte notes, int string, int notePosition)* – analogicznie do poprzedniej metody, *stopNote* wykonuje te same czynności z innymi parametrami w celu zatrzymania gry danej nuty.
- *onTouch* – metoda odpowiada za obsługę zdarzeń *MotionEvent* zachodzących na elementach szablonu graficznego z aktywowanym wcześniej *onTouchListener*. Zawiera instrukcję wyboru *switch*, która zależnie od *ID* naciśniętego przycisku uruchamia odpowiedni *case*. Jeśli odebrana zostane akcja odpowiadająca naciśnięciu danego pola, zostaje wywołana metoda *playNote* dla nuty jemu odpowiadającej. Jeśli natomiast akcja będzie oznaczała podniesienie palca z danego pola, analogicznie zostanie wywołana metoda *stopNote*. Metoda *onTouch* symuluje również zachowanie struny prawdziwej gitary uniemożliwiając grę dwóch nut z tej samej struny jednocześnie. Niemożliwe jest także zagranie nuty w przypadku gdy odtwarzana jest inna nuta z tej samej struny, ale o wyższym progu. Funkcjonalność ta jest możliwa do zaimplementowania dzieki tablicy *isNotePlaying*. Metoda *onTouch* obsługuje także naciśnięcie przycisku

powrotu, w przypadku którego wywoływana jest metoda *finish* kończąca pracę *Aktywności* oraz powrót do menu głównego.

- *Chords* – klasa pełni funkcję nauki akordów. Wykorzystywana jest w niej bilblioteka *MidiDriver*. Zawiera w sobie funkcję gry dowolnej, do której dołączony jest *Navigation Drawer* wysuwany z lewej strony, pozwalający na wybranie chwytów do nauki. Klasa zawiera następujące pola:
    - *midiDriver, event, notes, notesButtons, notesButtonsClick, isNotePlaying* – pola te pełnią identyczne role jak w klasie *Neck*. Tablica *notesButtonsClick* przechowuje jednak zmienne typu *TextView*, aby umożliwić umieszczenie na nich tesktu.
    - *TextView[] keysTexts* – tablica zawierająca w sobie elementy interfejsu odpowiadające za wyświetlanie nazw tonacji.
    - *TextView[] chordsTexts* – dwuwymiarowa tablica, która zawiera elementy interfejsu służące do wyświetlania nazw akordów.
    - *TextView[] chordsTextsFlats* – tablica odpowiedzialna za wyświetlanie znaku *bemol* przy odpowiednich akordach. Posiadają one osobną tablicę, ponieważ w formacie *Unicode* renderowane są z przerwą o długości jednej spacji przed i po znaku. W połączeniu z innym tesktem oznacza to uzyskanie niepożądanego wyglądu nazw akordów. Poprzez zastosowanie osobnych elementów *TextView* dla *bemoli* oraz odpowiednie ich umieszczenie względem pozostałej części nazw chwytów jesteśmy w stanie zlikwidować efekt niepotrzebnych spacji. Ponieważ nie wszystkie tonacje mają jednakową ilość dostępnych chwytów, oraz nie wszystkie akordy zapisane są z *bemolami* w nazwie, tablice *chordsTexts* oraz *chordsTextsFlats* nie posiadają wszystkich pól zainicjowalizowanych.
    - *ArrayList<ImageView> lastNoteButtonClickForInvisible* – lista w której umieszczane są elementy *notesButtonsClick* przeznaczone do zmiany ich parametru widoczności na *Invisible*.
    - *ArrayList<TextView> lastKeyForDrawableChange* – do listy dodawane są elementy interfejsu odpowiadające za wyświetlanie nazw akordów, przeznaczone do zmiany ich tła na domyślne.
    - *ArrayList<TextView> lastChordsForDrawableChange* – pełni taką samą rolę jak poprzednie pole, lecz dla elementów służących do wyświetlania nazw tonacji.
    - *ArrayList<TextView> lastChordsForGone* – lista, wkórej umieszczane są elementy interfejsu pełniące identyczną funkcję jak w poprzednim polu, przeznaczone do zmiany ich parametru widoczności na *Gone*.

- *boolean isDrawerOpen* – zawiera informacje o tym, czy element *Navigation Drawer* jest wysunięty.

Klasa zawiera następujące metody:

- *selectInstrument(int instrument)* – pełni identyczną rolę jak w klasie *Neck*.
- *onCreate(Bundle saveInstanceState)* – spełnia analogiczne funkcje jak w klasie *Neck*. Dodatkowo, zależnie od aktywnego języka systemowego, inicjalizowane są elementy szablonu graficznego odpowiedzialne za wyświetlanie tonacji i akordów H jako „H" lub „B". W Polsce przyjęło się używać pierwszej opcji, natomiast w krajach anglojęzycznych wykorzystywna jest opcja druga.
- *onResume()* – pełni identyczną rolę jak w klasie *Neck*, oraz jeśli Navigation *Drawer* nie jest otwarty, wywołuje metodę *openDrawerOnCreate*.
- *openDraweronCreate()* – metoda otwiera Navigation *Drawer* i odpowiednio zmienia wartość *isDrawerOpen*.
- *onPause, sendPlayNote, sendStopNote, playNote, stopNote* – metody te pełnią identyczną rolę jak w klasie *Neck*.
- *onTouch(View v, MotionEvent event)* – dla elementów szablonu graficznego dostępnych także w klasie *Neck* metoda pełni identyczną funkcję, podczas gdy Navigation *Drawer* nie jest otwarty. W przeciwnym wypadku funkcjonalność ta jest nieaktywna. Wyjątkiem jest element przycisku powrotu, który może zostać naciśnięty zawsze.
- *playChord(TextView chordsTexts, byte[][] chord, String[] fingers)* – metoda analogicznie do *playNote* jest odpowiedzialna za zagranie akordu oraz obsługę wszystkich zdarzeń z tym związanych:
    - Zmienione zostaje tło ostatniego wybranego chwytu dostępnego z *lastChordForDrawableChange*.
    - Widoczność elementów z tablicy *lastNoteButtonClickForInvisible* zostaje zmieniona na *Invisible*.
    - Tablice te zostają wyczyszczone oraz uzupełniane elementami odpowiadającymi wybranemu akordowi.
    - Dla wybranego chwytu zostaje zmienione tło *TextView* informujące użytkownika o aktualnie widocznym akordzie.
    - Podobnie jak w *playNote* zagrane zostają nuty należące do wybranego chwytu oraz uruchomione zostają związane z nimi animacje.
    - Dla każdej nuty pojawia się informacja, którym palcem należy ją zagrać, pobierana z parametru *fingers*.

- *changeKey(TextView keysTexts, TextView[][] chordsTexts, TextView[][] chordsTextsFlats, int key, int numberOfChords, numberOfFlats)* – metoda odpowieda za zmianę tła *TextView* aktualnie wybranej tonacji, analogicznie do zmian akordów z funkcji *playChord*, oraz wyświetla chwyty należące do niej. Widoczność akordów z poprzedniej wybranej tonacji zostaje zmieniona na *Gone* z użyciem tablicy *lastChordForGone*.
- *onClick(View v)* – metoda powiązana jest z elementami szablonu graficznego odpowiedającymi nazwom tonacji oraz chwytów. Zawiera instrukcję wyboru *switch*, która zależnie od parametru *ID* naciśniętego pola na ekranie powoduje wywołanie metody *changeKey* lub *playChord* dla wybranej tonacji i akordu. Chwyty zdefiniowane są w każdym *case* jako tablica dwuwymiarowa zmiennych typu *byte*, które odpowiadają odpowiednim pozycjom nut należącym do akordu na gryfie gitary.
- *onBackPressed()* – jeśli *Navigation Drawer* jest otwarty gdy zostanie naciśnięte pole wstecz na pasku nawigacji, zostaje on przed wyjściem zamknięty.

- *MidiUtils* – klasa abstrakcyjna zawierająca metody do obsługi plików *MIDI*, wykorzystywane przez klasę *Songs*. Wykorzystuje bibilotekę *Android MIDI Library*. Klasa zawiera jedno pole typu *String* o nazwie *sdPath*, ze ścieżką do folderu *Synthesian* tworzonego w pamięci wewnętrznej urządzenia. W *MidiUtils* znajdują się następujące metody:
  - dwa wariany metody *loadMIDI*:
    - *MidiFile loadMIDI(String fileName)* – zwraca obiekt typu *MidiFile* zależny od parametru *fileName* oznaczającego nazwę pliku MIDI.
    - *MidiFile loadMIDI()* – zwraca obiekt typu *MidiFile*, który odpowiada plikowi *_temp.mid* z folderu *Synthesian*.
  - *MediaPlayer loadMIDIForMediaPlayer(String fileName)* – wykorzystywana jest do załadowania pliku *MIDI* do odtwarzania z użyciem klasy *MediaPlayer*.
  - *saveMIDI(MidiFile midi, String fileName)* – metoda jako argument pobiera obiekt *midi*, który zapisuje do pamięci wewnętrznej wywołując na nim metodę *writeToFile*, dostępną z biblioteki *Android MIDI Library*.
  - *ArrayList<Long> getNoteOnTicks(MidiFile mf)* – dla każdego wydarzenia *NoteOn* z pierwszej ścieżki obiektu *mf* metoda pobiera wartość momentu jego startu, wyrażoną w cykach, oraz dodaje ją do *ArrayList ticks*, która jest zwracana na końcu metody.
  - *float getBPM(MidiFile mf)* – wyszukuje wydarzenie *TempoChange* w pierszej ścieżce obiektu *mf* oraz zwraca wartość *BPM* w nim zapisanej.

- Metody *ArrayList<Byte> getNotes(MidiFile mf), int getResolution(MidiFile mf), TimeSignature getTimeSignature(MidiFile mf), Tempo getTempo(MidiFile mf), int getTimeSignatureNumerator(MidiFile mf) oraz int getTimeSignatureDenominator(MidiFile mf)* działają na analogicznej zasadzie jak *getBPM*, zwracana jedynie jest inna wartość bądź obiekt:
  - *getNotes* - zwraca wartości nut, inaczej wysokości nut, przypisane do wydarzeń *NoteOn*.
  - *getResolution* - zwraca wartość *PPQ* zapisaną w obiekcie *mf*.
  - *getTimeSignature* - wyszukuje wydarzenie *TimeSignature* oraz je zwraca.
  - *getTempo* - wartością zwracaną jest wydarzenie *TempoChange*.
  - *getTimeSignatureNumerator* – zwraca licznik metrum przypisanego do wydarzenia *TimeSignature*.
  - *getTimeSignatureDenominator* – zwraca mianownik metrum z wydarzenia *TimeSignature*.

- *String prepareTempMidi(String filename, int BPM)* – metoda odpowiedzialna jest za utworznie pliku *_temp.mid*, który jest jedynym plikiem odtwarzanym przez aplikację. Wykonywane jest to w następujących krokach:
  - Wywoływana jest metoda *loadMIDI* z wykorzystaniem parametru *filename.*
  - Następnie dla każdego wydarzenia *NoteOn* z każdej ścieżki załadowanego obiektu *MidiFile* sprawdzana jest wysokość nuty w nim zapisana. Jeśli wartość ta odpowiada nucie niemożliwej do zagrania z wykorzystaniem dostępnej części gryfu zwracany jest *String „incompatibleNotes”* i metoda kończy działanie.
  - Każde wydarzenie ze ścieżek innych niż pierwsza, jeśli takie istnieją, jest do niej dodawane.
  - Sprawdzana jest wartość *PPQ* pliku. Jeśli wynosi ona 0 metoda zwraca *String „resolutionis0”* oznaczający, że plik *MIDI* jest niekompatybilny z aplikacją. Wartość *PPQ* musi być różna od zera aby w dalszej części metody nie doszło do dzielenia przez 0.
  - Wszystkie wydarzenia *NoteOn, NoteOff* oraz *TempoChange* ze ścieżki pierwszej są zapisywane do odpowiednich *ArrayList*. Następnie są one z niej usuwane.
  - Zapisane wydarzenia *NoteOn* są dodawane spowrotem do ścieżki z wartością odpowiadającą za ich start wyliczoną według formuły:

*(noteTick / (resolution / 480)) + ((resolution / (resolution / 480)) * numerator)*

gdzie:

- *noteTick* to obecna moment startu danej nuty.
- *resolution* to wartość *PPQ* zapisana w pliku *MIDI*.
- *numerator* to licznik metrum uzyskany przez wywołanie metody *getTimeSignatureNumerator*.
- 480 to nowa wartość *PPQ* ustalana w dalszej części metody.

Zastosowanie takiego działania do obliczenia cyków pozwala na uzyskanie ujednoliconych wartości dla każdego pliki *MIDI*, niezależnych od przypisanego *PPQ*. Jest to szczególnie ważne w kontekście obliczania opóźnień animacji dla każdej nuty w klasie *Songs*, które wykorzystują do tego wartości rozpoczęcia *NoteOn* im odpowiadające. Tempo odtwarzania jest niezależne od *PPQ*, które decyduje jedynie o długości ćwierćnuty w utworze, wyrażonej w cykach. Przykładowo, dla *PPQ* równego 920, ćwierćnuta trwałaby równą jemu ilość cyków, ósemka 480, półnuta 1840 itd. Aby zatem animacje gry poszczególnych nut były zsynchronizowane z odtwarzaniem utworu konieczne jest by plik _temp*.mid* posiadał zawsze taką samą wartość *PPQ*. Jest ona ustalana na 480, ponieważ zapewnia to utworzenie płynnych animacji nawet dla najkrótszych nut. Druga część wyrażenia dodaje do pliku pusty takt, by użytkownik miał czas na przygotowanie się do gry.

W dalszej części metody:

- analogicznie dodawane są wydarzenia *NoteOff*. Od momentu ich statu odejmowane jest jednak 10 cyków. Dzięki temu w przypadku gdy przykładowo nuta D3 kończy się w tym samym momencie, w którym zaczyna się nowa nuta D3, 10 cyków jest wystarczającym czasem by zapewnić płynne przejście między ich animacjami.
- wartość *BPM* jest zapisywana zgodnie z pobranym parametrem.
- wartość *PPQ* ustalona zostaje na 480.
- utworzony plik jest zapisywany w pamięci wewnętrznej urządzenia, w folderze Synthesian jako *_temp.mid*, a metoda zwraca *null*. W przypadku gdy plik już istnieje, jest on nadpisywany.

- *playTempMidi()* – metoda wykorzystuje statyczne pole *playerTemp* typu *MediaPlayer* z klasy *Songs*, do którego przypisuje obiekt otrzymany poprzez użycie metody *loadMIDIForMediaPlayer(„_temp.mid”)*. Na *playerTemp* wywoływana jest następnie metoda *start()* i utwór zaczyna być odtwarzany.
- *stopTempMidi()* – zasada jej działania jest analogiczna do *playTempMidi()*, lecz odtwarzanie utworu jest zatrzymywane dzięki metodzie *stop()*.

- *prepareMetronome()* – odpowiada za przygotowanie pliku *_metronome.mid*, który odtwarzany jest jeśli użytkownik wybierze opcję nauki z metronomem. Działanie metody przedstawia się w następujący sposób:
  - Ładowany jest plik *_temp.mid* z użyciem metody *loadMIDI()*.
  - Tworzony jest nowy obiekt *MidiFile*, do którego dodawane jest wydarzenie *ProgramChange* zmieniające instrument na metronom.
  - Do ścieżki dodawane są wydarzenia *NoteOn* z wartościami ich rozpoczęć równymi zmiennej *i* wyliczonej z użyciem formuły:

*i = i + (mf.getResolution() * 4 / getTimeSignatureDenominator(mf))*

gdzie:

- *getResolution* – zwraca *PPQ* obiektu *mf*.
- *getTimeSignatureDenominator* – dostarcza mianownik metrum wydarzenia *TimeSignature* obiektu *mf*.

  - Plik jest następnie zapisywany w pamięci wewnętrznej telefonu, w folderze *Synthesian* jako *_metronome.mid*.

- Metody *playMetronome* oraz *stopMetronome* działają identycznie jak *playTempMidi* i *stopTempMidi* lecz obsługują plik *_metronome.mid*

- *Songs* – klasa pełniąca funkcję nauki utworów. Wykorzystywana jest w niej bilblioteka *MidiDriver* oraz *Android MIDI Library*. Klasa zawiera w sobie funkcjonalność dostępną z poziomu gry dowolnej, do której dołączony jest *Navigation Drawer* wysuwany z lewej strony, pozwalający na wybranie utworów do nauki. *Songs* posiada następujące pola:
  - *midiDriver, event, notes, notesButtons, notesButtonsClick, isNotePlaying* - pola te pełnią identyczną role jak w klasie *Neck*.
  - *ImageView[][] notesSongOuter* – przechowuje elementy interfejsu będące białymi pierścieniami, wyświetlanymi dla każdej nuty utworu w trybie nauki.
  - *ImageView[][] notesSongInner* – zawiera elementy interfejsu będące białymi kołami wypełniającymi wcześniej wspomniane pierścienie, wyświetlane dla każdej nuty utworu.
  - *ImageView[] notesSongEmpty* – pełni rolę identyczną jak *notesSongInner* dla pól na ekranie odpowiadającym pustym stronom.

- *Button soundRadio, TextView tempoBPM, Button tempoPlus, Button tempoMinus, Button metronomeRadio* – odpowiadają elementom szablonu graficznego widocznym gdy otwarty jest Navigation *Drawer*.
- *ArrayList<TextView> songs* – zawiera elementy pełniące funkcję wyświetlania nazw utworów w *Navigation Drawer*.
- *int lastSongId* – przechowuje wartość *ID* ostatniej odtwarzanej piosenki.
- *int BPM* – przechowuje wartość *BPM* ustaloną przez użytkownika po wybraniu utworu.
- *boolean withSound* – zawiera informację o tym, że nauka danego utworu ma zostać rozpoczęta z jego odtworzeniem.
- *boolean withMetronome* – określa czy do nauki ma zostać użyta dodatkowo funkcja metronomu.
- *String chosenSong* – przechowuje nazwę pliku wybranego utworu do nauki.
- *static MediaPlayer playerTemp* – zmienna wykorzystywana do odtwarzania pliku *_temp.mid.*
- *static MediaPlayer playerMetronome* – zmienna wykorzystywana do odtwarzania pliku *_metronome.mid.*
- *AnimatorSet animSet1* – przechowuje animatory wykorzystywane do animacji pól odpowiadającym pustym strunom.
- *AniamtorSet animSet2* – przechowuje animatory używane do animacji pierścieni pól odpowiadającym nutom z progów gitary.
- *AnimatorSet animSet3* - przechowuje animatory używane do animacji wewnętrznych kół pól odpowiadającym nutom z progów gitary.
- *AnimatorSet arrowAnimatorSet* – zawiera animatory wykorzystywane do animacji strzałki informującej o kierunku bicia gitarowego akordu.
- *ImageButton informationButton* – przycisk używany do otworzenia okna informacji.
- *Button okButton* – przycisk „Ok” widoczny po przejściu do okna informacji.
- *DrawerLayout drawer* – zmienna wykorzystywana do kontroli zachowania *Navigation Drawer* obecnego w wykorzystywanym szablonie graficznym.
- *boolean isDrawerOpen* – informuje o tym czy *drawer* jest otwarty.

Klasa zawiera następujące metody:

- *selectInstrument(int instrument)* – pełni rolę identyczną jak w klasie *Neck*.
- *onCreate(Bundle saveInstanceState)* – metoda wywoływana jednokrotnie podczas uruchamiania Aktywności, która posiada funkcje takie same jak odpowiadająca jej metoda z klasy *Neck*, oraz dodatkowo:

- Pola *informationButton, okButton, soundRadio, tempoBPM, tempoPlus, tempoMinus, metronomeRadio* kojarzy z odpowiednimi widokami szablonu graficznego oraz przypisuje im *onTouchListener*.
- Inicjalizuje pola typu *AnimatorSet*.
- Wywołuje metodę *addSongsToMenu()*.
- Jeśli *ArrayList songs* nie jest pusta, zmiennym *chosenSong, BPM* oraz tempo*BPM* zostają przypisane odpowiednie wartości pobrane z pierwszego utworu w liście.
- Do każdego elementu listy *songs* dodawany jest *onTouchListener*.
- Zmiennej *drawer* przypisywany zostaje *DrawerListener* z zaimplementowaną metodą *onDrawerOpened*. Gdy *drawer* zostanie otworzony oraz jeśli lista *songs* nie jest pusta, wywoływane zostają metody *stopTempMidi()* i *stopMetronome()* z klasy *MidiUtils*. Zatrzymywane jest także działanie zbiorów animatorów *animSet1, animSet2, animSet3 i arrowAnimatorSet*.

- *onResume()* – pełni funkcję identyczną jak w klasie *Neck*.
- *openDrawerOnCreate()* - pełni funkcję identyczną jak w klasie *Chords*.
- *onPause()* - pełni funkcję identyczną jak w klasie *Neck*.
- Metody *sendPlayNote, sendStopNote, playNote, stopNote* – posiadają takie same funkcje jak w klasie *Neck*.
- *onTouch(View v, MotionEvent event)* - dla elementów szablonu graficznego dostępnych także w klasie *Neck* metoda pełni funkcję gry dowolnej, podczas gdy *drawer* nie jest otwarty. W przeciwnym wypadku funkcjonalność ta jest nieaktywna. Naciśnięcie pozostałych elementów z aktywnym *onTouchListener* powoduje:
  - przycisk powrotu - jeśli lista *songs* nie jest pusta, wywołanie metod *stopTempMidi()* oraz *stopMetronome()*, jak również wywołanie metody *release()* na *playerTemp* i *playerMetronome*, co powoduje ich zwolnienie z pamięci operacyjnej urządzenia. Niezależnie od zawartości *songs* następuje wyjście z *Aktywności*.
  - *informationButton* – zmianę wartości widoczności okna informacji, tekstu informacji oraz przycisku *okButton* na *Visible*, jak również uruchomienie animacji ich pojawania się na ekranie.
  - *okButton* – rozpoczęcie animacji zmieniających przezroczystość tekstu, tła informacji oraz przycisku *okButton* na wartość od 1 do 0. Dodatkowo po zakończeniu animacji jeog widoczność jest zmieniana na *Gone*, by nie kolidował on z przyciskiem *Play*, który częściowo

zajmuje ten sam obszar ekranu. W przeciwnym wypadku odtworzenie utworu powodowałoby także uruchomienie wcześniej wymienionych animacji.

- *Play* – ukrycie wysuwanego menu. Po upływie 500 milisekund wywołana zostaje metoda *prepareTempMidi(chosenSong, BPM)*. Bez tego opóźnienia zamykanie okna *drawer* mogłoby zachodzić z zacięciami. Jeśli *prepareTempMidi* zwróci *String* różny od *null*, to analogicznie do *case informationButton* pokazywana jest informacja o błędzie, gdzie tekst informacji zależy od wartości zwróconej przez *prepareTempMidi* i jest pobierany z zasobów *string.xml*. Działanie metody zostaje po tym przerywane. W przypadku gdy *String* wynosi jednak *null*, wywoływana jest metoda *showSong()*. Uruchomione mogą również zostać *playTempMidi()*, *prepareMetronome()* i *playMetronome()* z klasy *MidiUtils*, zależnie od wartości *withSong* oraz *withMidi*.
- *SoundRadio* – zmianę wartości *withSound* oraz tła *soundRadio*.
- *MetronomeRadio* – zmianę wartości *withMetronome* oraz tła *metronomeRadio* analogicznie do *case SoundRadio*.
- *TempoPlus* – zwiększenie wartości *BPM* o 5 oraz uaktualnienie tekstu *tempoBPM*.
- *TempoMinus* – jeśli wartość *BPM* jest większa lub równa 6, zmiejszenie jej *o 5* oraz uaktualnienie tekstu *tempoBPM*.
- Dla każdego elementu *songs*:
  - zmianę tła ostatnio wybranego utworu na domyślne wykorzystując *lastSongId.*
  - przypisanie odpowiednich wartości zmiennym *chosenSong, BPM* oraz *tempoBPM* zależnym od wybranego utworu.
  - zmianę tła wybranego utworu na podświetlone.
  - przypisanie *ID* wybranego utworu do *lastSongId*.
  - odświeżenie zbiorów animatorów *animSet1, animSet2, animSet3*, oraz *arrowAnimatorSet* by nie zawierały elementów wygenerowanych na potrzeby poprzednio wybranego utworu.

- *addSongToMenu()* – metoda odpowiada za dodanie plików z folderu *raw* do folderu *Synthesian*, oraz przeskanowanie tego katalogu i umieszczenie zawartych w nim utworów do *drawer*. Zostaje to osiągnięte w następujących krokach:

        - Dla każdego pliku w folderze *raw* tworzony jest *InputStream*.
        - Następnie w tej samej pętli tworzony jest *OutputStream*, który przygotowuje jego zawartość do zapisu z odpowiednią nazwą pobraną z zasobów *strings.xml*, zależną od języka systemowego.
        - W folderze *Synthesian* powstają w ten sposób pliki *z* rozszerzeniem *.mid* przygotowane na potrzeby aplikacji.
        - Dla każdego pliku *MIDI* znalezionego w tym katalogu tworzony jest *TextView*, który nasępnie zostaje dodany do listy *songs* oraz menu utworów w *drawer*.
    - *String removeExtension(String s)* – metoda usuwa ze zmiennej *s* rozszerzenie ".mid".
    - *float convertDpToPixel(float dp, Context context)* – zamienia wartość *dp (ang. Density-independent Pixel)* na jej odpowiednik wyrażony w pikselach.
    - *float ticksToMs(long ticks, int BPM, int resolution)* – metoda wykorzystywana jest do zamiany cyków na wartość wyrażoną w milisekundach. Formuła tego działania wygląda następująco:

      *ticks * (float) 60000 / (BPM * resolution)*

      gdzie:
        - *ticks* – wartość którą chcemy zamienić.
        - *BPM* – *BPM* utworu, z którego pochodzi liczba *ticks*.
        - *resolution* – wartość *PPQ* utworu z którego została pobrana wartość *ticks.*

    - *showSong()* – metoda odpowiedzialna za wyświetlenie animacji informujących użytkownika o tym, kiedy należy nuty zawarte w wybranym utworze zagrać. Jeśli fragmenty piosenki należy zagrać z wykorzystaniem bicia gitarowego, wyświetlane są także strzałki podpowiadające jego kierunek. Zostaje to osiągnięte w następujących krokach:
        - Zamykany jest *drawer*.
        - Tworzone są:
            - *ArrayList<Long> ticksOn* zawierająca wartości pobrane z metody *getNoteOnTicks*.
            - *ArrayList<Byte> notesValues* wypełniona wysokościami każdej nuty w utworze.
            - *HashMap<Byte, Long> previousNote*, która dla każdej wysokości nuty występującej w utworze, przypisuje cyk

ostatniego wydarzenia *NoteOn* z identyczną wartością odpowiedającą wysokości.

- *LinkedList<Long> notesForChords* – zbiera nuty, które należy zagrać biciem gitarowym.

- Dla każdego wydarzenia *NoteOn* obliczana jest wartość *animationDuration* wyrażona w cykach:
  - Początkowo jest ona ustalana na wartość równą 820, która została dobrana tak, by pojawiające się animacje nie trwały zbyt długo lub krótko.
  - Jeśli nuta o danej wysokości pojawiła się już w utworze, to sprawdzany jest następujący warunek:

(ticksOn.get(i) – previousNotes.get(previousNoteValue)) < animationDuration

gdzie:

- *ticksOn.get(i)* – moment startu danego wydarzenia.
- *previousNotes*.get(previousNoteValue) – moment startu ostatniej nuty o tej samej wysokości.

Jeśli wynik tego odejmowania okaże się być mniejszy od *animationDuration* konieczna jest zmiana wartości tej zmiennej. Obliczana jest ona w ten sposób:

animationDuration = (ticksOn.get(i) – previousNotes.get(previousNoteValue)) – 100

Od wyniku pierwszego działania odjemowane jest jeszcze 100 cyków, ponieważ tyle trwa animacja znikania elementów szablonu podpowiadających, którą nutę zagrać. Dzięki temu niemalże w każdym przypadku animacja będzię kończyć się zanim nastąpi rozpoczęcie animacji dla kolejnej nuty. Jeśli jednak te same nuty będą występowały bardzo szybko po sobie *animationDuration* moze wynieść mniej niż 0. Wartość tej zmiennej jest wtedy ustalana na 0.

- Metoda wypełnia *notesForChords* i odtwarza animacje strzałek bicia gitarowego w następujący sposób:
  - Jeśli spełniony zostanie jeden z warunków:
    - Moment rozpoczęcia obecnej nuty pomniejszona o wartość jej odpowiadającą z nuty poprzedniej jest wystarczająco duża.

- Są one jednakowe.
- Sprawdzana nuta jest ostatnia w utworze.

to gdy w *notesForChords* znajduje się więcej niż 3 elementy, wyświetlane są animacje strzałek. Ich kierunek oraz zasięg ustalany jest na podstawie wysokości nut.

- Lista *notesForChords* zostaje czyszczona, oraz dodawany do niej jest moment startu aktualnie sprawdzanego wydarzenia *NoteOn*.

- Dla każdej nuty w utworze tworzone są animatory elementów szablonu pełniące funkcję podpowiedzi do jej zagrania.
- Ponieważ nutę H3 można zagrać wykorzystując czwarty próg gitary lub pustą strunę H, co jest łatwiejsze, do animowania wybierana jest zawsze druga opcja.
- Utworzone animatory posiadają długość równą *animationDuration* dla danej nuty, oraz opóźnienie równe wartości momentu startu danego wydarzenia *NoteOn* pomniejszone o *animationDuration*. Wyrażane są one w milisekundach po zamianie z cyków przy użyciu metody *ticksToMs*.
- Po umieszczeniu wszystkich animatorów w odpowiednich zbiorach są one uruchamiane.

- *onBackPressed()* – działanie metody jest identyczne jak w klasie *Chords*.

c. Cechy charakterystyczne interfejsu

Interfejs aplikacji został zaprojektowany tak, by umożliwić intuicyjne i płynne poruszanie się po aplikacji. Zostało to osiągnięte poprzez:

- zastosowanie minimalistycznego designu. W każdym momencie korzystania z aplikacji brak jest elementów, z którymi nie można wejść w interakcję. Wyjątek stanowi menu główne, gdzie logo[22] pełni funkcję wyłącznie ozdobną.
- zapewnienie, że każda obsługiwana interakcja użytkownika z aplikacją dostarczy mu graficzną informację zwrotną. Przykładowo, naciśnięcie przycisku przejścia do gry dowolnej powoduje zmianę jego tła białe.

- wykorzystanie *Navigation Drawer* w jednakowy sposób dla trybów nauki chwytów oraz nauki piosenek.

Główną częścią interfejsu jest gryf gitary[23], widoczny w każdem momencie korzystania z aplikacji poprzez zastosowanie przezroczystości większości pozostałych elementów. Możliwe dzięki temu jest uzyskanie efektu płynnego przejścia z menu głównego do dowolnego trybu nauki.

5. Użytkowanie aplikacji

a. Wymagania systemowe oraz sprzętowe

Aplikacja przeznaczona jest na systemy *Android* w wersji od 4.1 *(Jelly Bean)*, zaprezentowanej w czerwcu 2012 roku, do 8.1 *(Oreo)*, która jest aktualnie wersją najnowszą. Z tego względu zalecane jest korzystanie z urządzenia mobilnego oficjalnie je wspierającego.

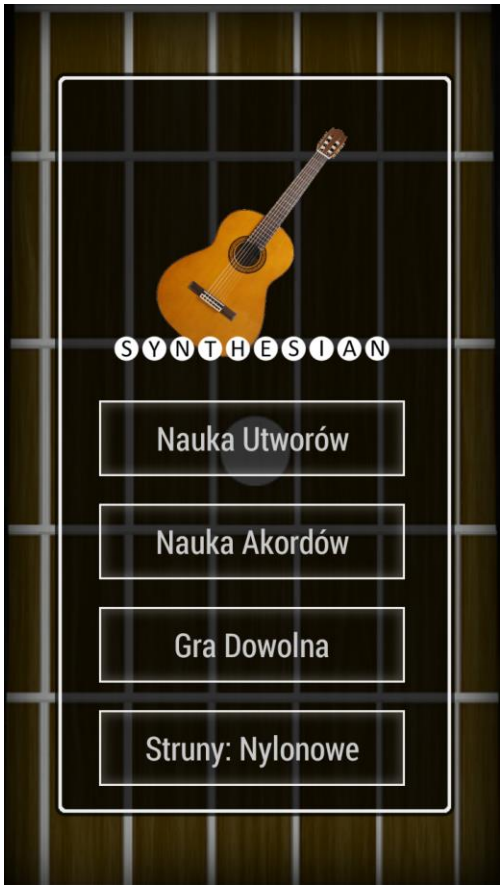
Podczas instalacji aplikacja poprosi użytkownika o zaakceptowanie pozwoleń na zapis oraz odczyt plików z pamięci wewnętrznej. Funkcje te są wykorzystywane do przygotowania *_temp.mid* i *_metronome.mid* oraz odczytywania plików z folderu *Synthesian*.

b. Przedstawienie działania aplikacji

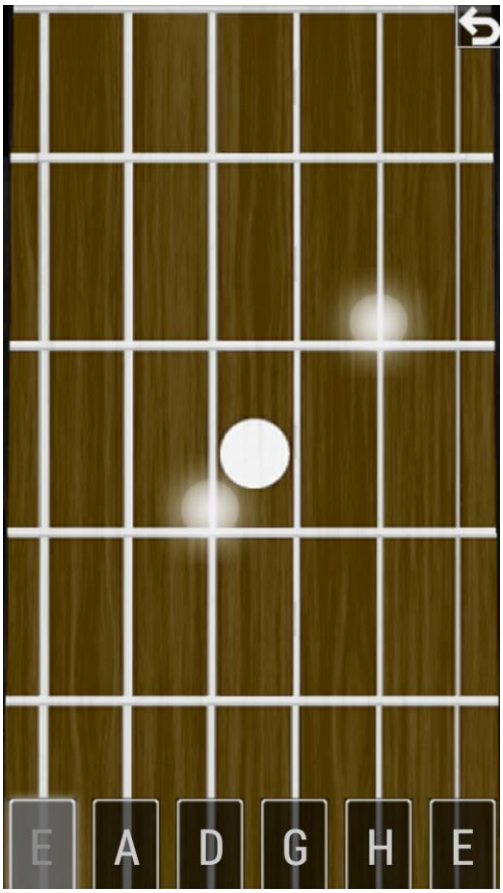
Aplikacja uruchamiana jest poprzez naciśnięcie jej ikony [Rys. 8].

Rys. 8 Ikona aplikacji

Po uruchomieniu widoczny jest ekran startowy z logiem aplikacji. Po 2-3 sekundach następuje przejście do menu głównego [Rys. 9].



Rys. 9 Menu główne aplikacji



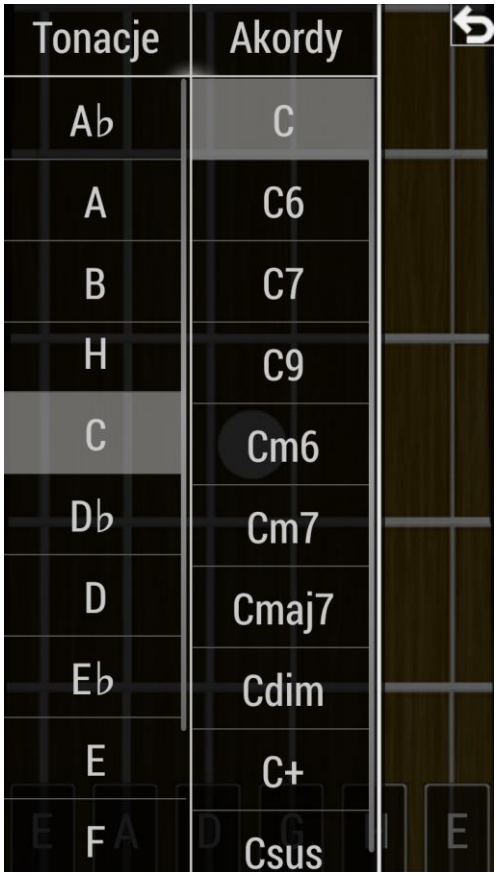
Rys. 10 Tryb gry dowolnej

Z jego poziomu mamy dostęp do każdej funkcji aplikacji przedstawionej w wymaganiach funkcjonalnych wykorzystując przyciski:

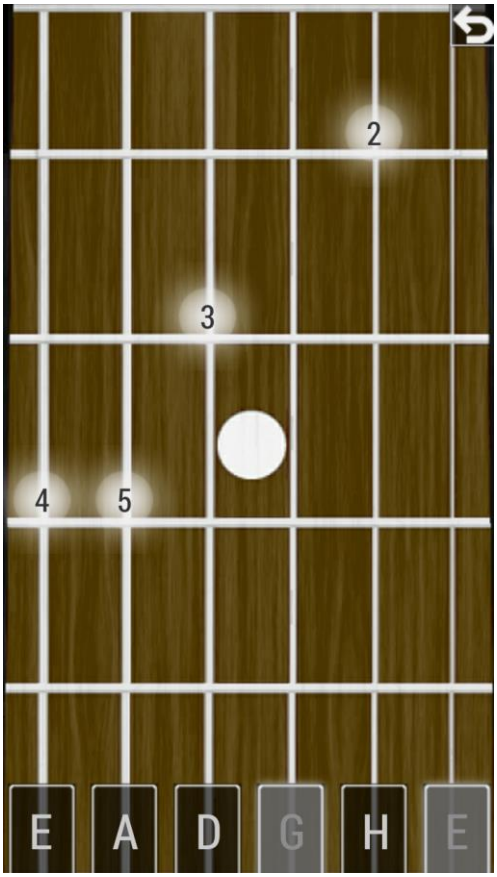
- „Struny: Nylonowe” – otwiera podmenu z wyborem dźwięku gitary. Możliwe opcje to „Nylonowe” oraz „Stalowe”.
- „Gra Dowolna” – umożliwia przejście do trybu gry dowolnej.
- „Nauka Akordów” – uruchamia tryb nauki chwytów.
- „Nauka Utworów” – umożliwia przejście do trybu nauki piosenek.

W trybie gry dowolnej użytkownik może naciskać odpowiednie pola na gryfie, co powoduje odtworzenie nut im odpowiadających [Rys. 10]. U dołu ekranu umieszczone są także pola odpowiadające uderzeniu pustych strun. Możliwe jest zagranie maksymalnie sześciu nut jednocześnie – po jednej z każdej struny. W prawym górnym rogu ekranu widoczny jest przycisk, którego naciśnięcie powoduje powrót do menu głównego.

W trybie nauki akordów użytkownik z wysuwanego menu wybiera tonację oraz interesujący go chwyt [Rys. 11]. Część z nich widoczna jest dopiero po przesunięciu podmenu, w których się znajdują. Po wybraniu chwytu zostaje on odtworzony oraz zaprezentowany na gryfie. Numery widoczne na niektórych polach odpowiadają palcom lewej ręki, którymi należy dane nuty nacisnąć [Rys. 12]. Po ponownym wysunięciu menu wyboru akordów ostatni wybór zostaje podświetlony, informując jaki chwyt jest obecnie prezentowany. Naciśnięcie strzałki powrotu powoduje przywrócenie menu głównego.



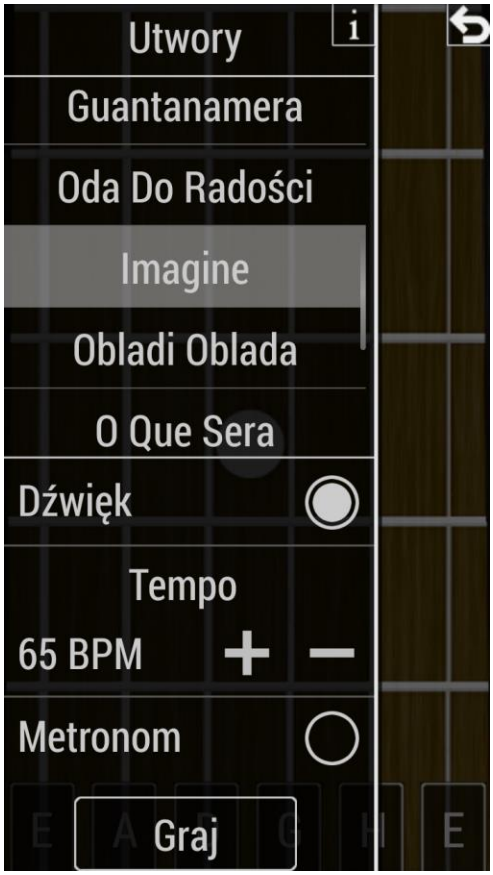
Rys. 11 Menu wyboru akordów



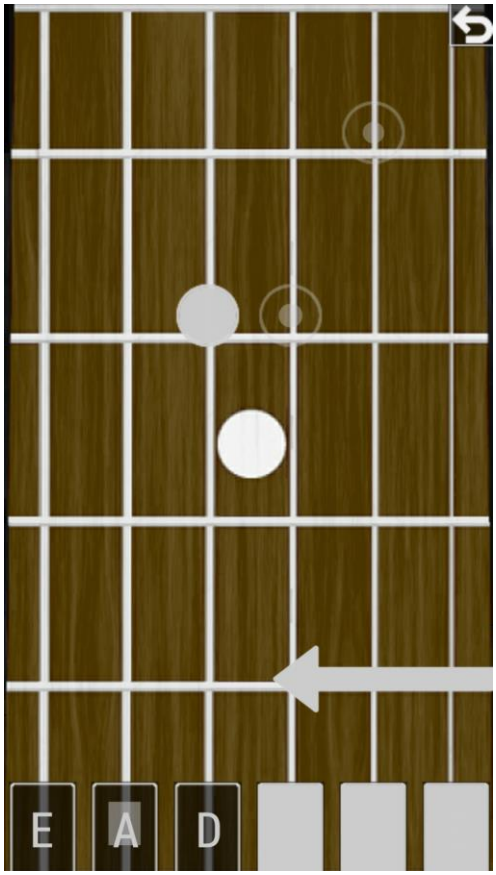
Rys. 12 Chwyt *C* zaprezentowany na gryfie

Tryb nauki utworów posiada wysuwane menu, w którym z przesuwanego podmenu wybieramy interesującą nas piosenkę [Rys. 13]. Przycisk „i” widoczny na górze powoduje pojawienie się informacji o możliwości importowania własnych plików *MIDI* do aplikacji, np. wykorzystując komputer stacjonarny. Przycisk „Dźwięk” umożliwia wybranie, czy podczas nauki w tle odtwarzany ma być wybrany utwór. Zakładka „Tempo” służy regulacji tempa nauki. Po wybraniu nowej piosenki wartość *BPM* zostaje automatycznie zmieniona. Przycisk „Metronom” pozwala na wybranie czy w nauce ma pomagać metronom wystukujący rytm.

Po naciśnięciu przycisku „Graj” menu się zamyka i zaczyna się gra utworu. Aplikacja daje użytkownikowi jeden pusty takt na przygotowanie. Nauka polega na grze nut odpowiadających pojawiającym się polom. Pola te, z początku puste, wypełniają się z upływem czasu [Rys. 14]. Jego ilość zależy od danej nuty oraz tempa odtwarzania. Gdy wypełnią się do końca jest to moment, w którym należy je nacisnąć. Niektóre nuty należy zagrać biciem gitarowym. W takim przypadku na ekranie pojawia się strzałka informująca o jego kierunku oraz jakie struny ma ono obejmować [Rys. 14]. Ponowne wysunięcie menu powoduje przerwanie odtwarzania utworu.



Rys. 13 Menu wyboru utworu



Rys. 14 Nauka utworu

6. Podsumowanie

a. Możliwości dalszego rozwoju

Wszystkie funkcje aplikacji spełniają przypisane im role, jednak możliwe jest wprowadzenie udoskonaleń poszerzających metody nauki. Funkcją, która znacznie wzbogaciłaby ich potencjalny rozwój niewątpliwie jest obsługa gitary elektryczno-akustycznej podłączonej do urządzenia mobilnego. Umożliwiłoby to zaimplementowanie reakcji aplikacji bezpośrednio na uderzenia strun z niej pochodzące. Zmniejszyłoby to oczywiście liczbę użytkówników, ponieważ wymagane byłoby posiadania odpowiedniego modelu gitary.

Jednak nawet pomijając tę funkcję możliwe jest zaimplementowanie dodatkowych usprawnień. Tryb gry dowolnej mógłby obsługiwać odtwarzanie dźwięków poprzez przesuwanie palca po ekranie, obecnie każde z pól należy nacisnąć z osobna. Nauka akordów skorzystałaby na dostepności większej ilości progów gryfu, dostępnych po jego przesunięciu. Umożliwiłoby to wprowadzenie większej ilości chwytów do aplikacji. Nauka utworów mógłaby prezentować każdą piosenkę w interwałach równych np. dwóm taktom. Dzięki temu użytkownik miałby szansę przećwiczenia piosenki kawałek po kawałku, przechodząc do następnej części dopiero po opanowaniu poprzedniej.

Aplikacja zaimplementowana została w sposób umożliwiający łatwe wykorzystanie niektórych metod do działania różnych funkcji. Przykładowo metoda *sendPlayNote()* używana jest zarówno przez *playNote()* jak i *playChord()*. Oznacza to potencjalne zmniejszenie czasu potrzebnego na dodanie nowych funkcji do apliakcji.

b. Wnioski

Wykorzystanie aplikacji mobilnej jest sposobem nauki gry na gitarze posiadającym swoje zalety oraz wady. Instalacja zajmuje zaledwie kilka minut a korzystanie z niej możliwe jest wszędzie gdzie zabierze się ze sobą smartfon czy tablet. W porównaniu do szkoły muzycznej aplikacja umożliwia swobodniejszą naukę, z tempem dopasowanym do własnych potrzeb. Pozwala ona także na kształcenie się jedynie w aspektach gry na gitarze interesujących danego użytkownika. Nauka w szkole muzycznej prowadzona jest z wykorzystaniem nut, których płynne czytanie wymaga kilka lat ćwiczenia. Aplikacja stworzona na potrzeby tego projektu umożliwia poznawanie chwytów i grę utworów bez znajomości nut lub tabulatury.

Z drugiej strony decydując się na samodzielną naukę z wykorzystaniem aplikacji nie mamy do naszej dyspozycji nauczyciela, którego wiedza i umiejętności wykraczają poza to co może umożliwić smartfon. Szkoła muzyczna rozwija nie tylko umiętności gry na gitarze, lecz również słuch czy poczucie rytmu. Również zakres wiedzy w niej dostarcznej jest znacznie obszerniejszy niż informacje możliwe do przekazania wykorzystując aplikację mobilną. Ukończenie takiej szkoły umożliwia uzyskanie dyplomu świadczącego o posiadanych kwalifikacjach co pozwala m. in. na ubieganie się o przyjęcie do akademii muzycznych.

Aplikacja mobilna jest zatem alternatywą, z której osoby nie zainteresowane nauką gry na gitarze w szkole muzycznej lub ognisku muzycznym, bądź nie mogące sobie na nią z różnych względów pozwolić, powinny spróbować. Rozwój technologiczny postępuje w dzisiejszych czasach na tyle szybko, że w najbliższych latach aplikacje prawdopodobnie umożliwiać będą ćwiczenie gry w znacznie bardziej efektywny sposób. Być może urządzenia mobilne znajdą zastosowanie również w szkołach muzycznych.

7. Bibliografia

[1] Bloch Joshua: *Effective Java*, 2001

[2] Deitel Harvey: *Android for Programmers: An App-Driven Approach*, 2011

[3] Eckel Bruce: *Thinking in Java*, 1998

[4] Friesen Jeff: *Learn Java for Android Development*, 2010

[5] Griffiths David, Griffiths Anthony J. F. : *Head First Android Development: A Brain-Friendly Guide*, 2015

[6] Hunt Charlie, John Binu: *Java Performance*, 2011

[7] Marchwica Wojciech (red.): *Słownik Muzyki*, 2006

[8] Martin Robert C. : *Clean Code. A Handbook of Agile Software Craftmanship*, 2008

[9] Meier Reto: *Professional Android 2, Application Development*, 2008

[10] Sierra Kathy, Bates Bert: *Java. Rusz głową!*, 2003

[11] http://www.mkidn.gov.pl/pages/strona-glowna/uczniowie-i-studenci/szkoly-artystyczne/system-ksztalcenia-artystycznego-stopnia-podstawowego-i-sredniego/ksztalcenie-muzyczne.php

[12] http://www.buk.gmina.pl/pl/ogniska-muzyczne.html

[13] http://www.muzyczna-zywiec.pl/ognisko-muzyczne,7.html

[14] http://www.ognisko-muzyczne.eu/

[15] https://pl.wikipedia.org/wiki/Gryf_(chordofony)

[16] https://pl.wikipedia.org/wiki/Tonacja

[17] https://developer.android.com/studio/index.html

[18] https://www.gimp.org/

[19] https://www.popsci.com/install-android-oreo-now

[20] https://github.com/billthefarmer/mididriver/

[21] https://github.com/LeffelMania/android-midi-lib/

[22] https://lodz.adwent.pl/guitar_png3374/

[23] https://www.shutterstock.com/pl/image-vector/guitar-fretboard-54840109?src=u2k0QE_mFZeB0Dg85ry8BQ-1-18