Dawid Dopart 248895
Błażej Ułanowicz 249452

Prowadzący:
dr inż. Dominik Żelazny

Urządzenia peryferyjne – laboratorium
Ćwiczenie 11 – obsługa karty muzycznej

1. Cel ćwiczenia

Celem ćwiczenia było utworzenie aplikacji, która pozwala obsługiwać działanie karty dźwiękowej. W ramach laboratorium zrealizowano poniższe funkcjonalności:

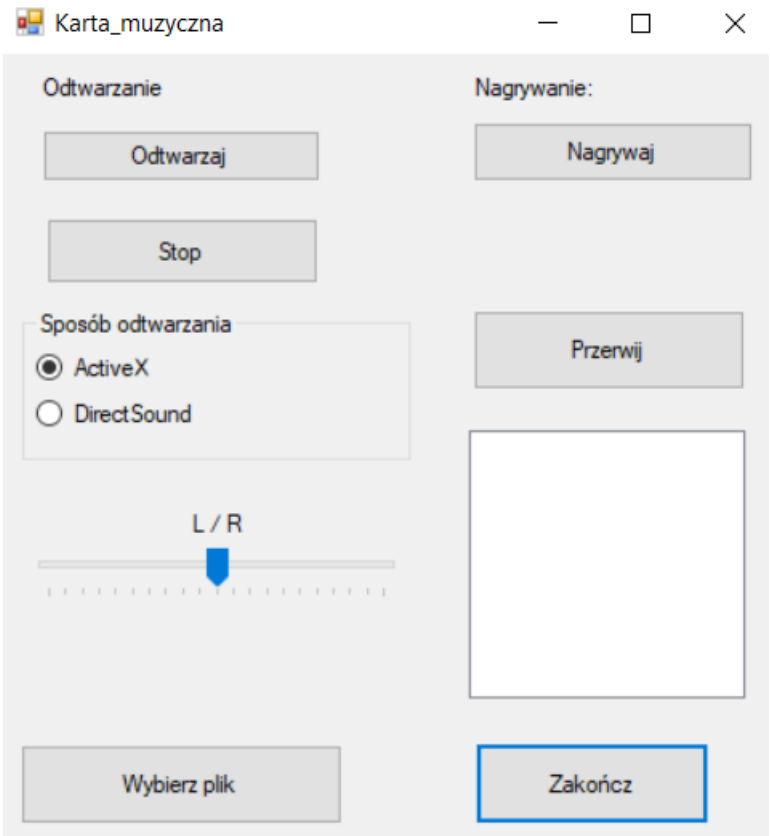
- Odtwarzanie dźwięku z plików WAV – ActiveX, DirectSound
- Sczytywanie nagłówków plików WAV
- Odtwarzanie plików mp3 - AxWindowsMediaPlayer
- Sczytywanie dźwięku z mikrofonu, a następnie zapis do pliku

2. Wykorzystane technologie i środowisko pracy

Implementację funkcjonalności z treści laboratorium wykonano w C#, a do obsługi karty muzycznej skorzystano z ActiveX, DirectSound oraz NAudio. Przy tworzeniu interfejsu użytkownika wykorzystano Windows Forms. Aplikacja testowana była na komputerze z Windows 10 z wykorzystaniem zintegrowanej karty dźwiękowej.

3. Opis działania

Na potrzeby działania aplikacji utworzono prosty graficzny UI, pozwalający korzystać z zaimplementowanych funkcji będących realizacją zadań zawartych w treści laboratorium.



Zrzut ekranu 1. GUI wykonanej aplikacji

3.1. Odtwarzanie dźwięku z plików WAV.

Do odtworzenia pliku WAV wykorzystano dwie technologie: ActiveX (z biblioteki *System.Media*) oraz, wchodzącą w skład DirectX, DirectSound.

Najprostszym sposobem na odtwarzanie dźwięku w formacie WAV jest skorzystanie z zestawu kontrolek ActiveX. Korzystając z atrybutów i metod klasy *System.Media.SoundPlayer* (obsługującej odtwarzacz) można najpierw wskazać umiejscowienie pliku WAV z dźwiękiem (atrybut *SoundLocation*), a następnie załadować ten plik, utworzyć nowy wątek i rozpocząć odtwarzanie dźwięku – wszystko za pomocą pojedynczej metody *Play()*. Warto zauważyć, że korzystamy tutaj właśnie z metody *Play()*, a nie *PlaySync()*, gdyż ta druga nie tworzy nowego wątku i zablokowałaby główny, co uniemożliwiłoby komunikację z aplikacją. Zatrzymanie odtwarzania jest tak samo proste jak jego rozpoczęcie, czyli wykorzystujemy metodę *Stop()*.

Do bardziej zaawansowanego odtwarzania dźwięku wykorzystana została technologia DirectSound (z biblioteki *Microsoft.Directx.DirectSound*). Pozwala ona nie tylko na podstawowe operacje, ale również na modyfikowanie takich parametrów jak panning czy głośność. Pracę rozpoczynamy od utworzenia instancji klasy *Device*, która odpowiada za urządzenie używane do odtwarzania dźwięku. Jako argument konstruktora podajemy *DSoundHelper.DefaultPlaybackDevice*, informując tym, że chcemy będziemy korzystać z domyślnego urządzenia odtwarzającego komputera. Ustawiamy następnie *CooperativeLevel* naszego urządzenia na *Normal* – jest to najbardziej wydajne ustawienie, jednakże blokowany jest format dźwięku na 44kHz, 16-bit oraz stereo. Następnie tworzymy bufor (*BufferDescription*), który służy do opisania dokładnych ustawień sesji odtwarzania, w tym, czy aplikacja może zmieniać jakieś ustawienie. W naszym wypadku chcemy, aby możliwa była zmiana ustawienia „panningu", więc *ControlPan* ustawiamy na *true*. Pozostało stworzyć bufor z dźwiękiem i do tego zostaje wykorzystana klasa *SecondaryBuffer*, do której konstruktora wstawiana jest ścieżka do żądanego pliku, opis odtwarzacza w postaci obiektu klasy *BufferDescription* oraz urządzenia dźwiękowego w postaci obiektu klasy *Device*. Kolejne operacje jak odtwarzanie czy zatrzymywanie wykonywane są za pomocą metod nowoutworzonego obiektu – *Play()*, *Stop()*, natomiast panning można zmieniać przypisując odpowiednią wartość do pola *Pan*.

```
soundDevice = new Microsoft.DirectX.DirectSound.Device(DSoundHelper.DefaultPlaybackDevice);
soundDevice.SetCooperativeLevel(parent,CooperativeLevel.Normal);
buffDesc = new BufferDescription();
buffDesc.ControlEffects = false;
buffDesc.ControlPan = true;
sound = null;
```

Kod 1: Inicjalizacji obiektów odpowiedzialnych za odtwarzanie (DirectSound)

```
case PlayerType.DIRECTSOUND:
    {
        sound = new SecondaryBuffer(wavFile, buffDesc, soundDevice);
        sound.Play(0, BufferPlayFlags.Default);
        break;
    }
```

Kod 2: Odtwarzanie dźwięku w przypadku DirectSound

3.2. Sczytywanie nagłówka pliku WAV

Format WAV jest to format plików dźwiękowych oparty o format RIFF. Poszerzony on został o takie pola jak informacje o strumieniu, kodek, częstotliwość próbkowania czy liczba kanałów. Dzięki znajomości przeznaczenia kolejnych bajtów nagłówka, podejrzenie tych informacji sprowadza się do manipulacji strumieniem bitowym. Korzystamy w tym przypadku z dwóch klas – *FileStream* do utworzenia strumienia danych z pliku oraz *BinaryReader* do przygotowania strumienia do odczytu bitów. Następnie ustawiamy nasz strumień na odpowiednią pozycję za pomocą parametru *Position* i sczytujemy interesujące nas dane. W tym przypadku sczytaliśmy cztery informacje: typ pliku (4 bajty zaczynając od 8-mego) – korzystamy tutaj z metody *ReadChar*, ponieważ informacją są cztery litery typu; liczba kanałów (2 bajty zaczynając od 22-ego); częstotliwość próbkowania (4 bajty zaczynając od 24-ego); liczba bitów przypadająca na próbkę (2 bajty zaczynając od 34-ego).

```
public Dictionary<String, String> readLoadedWAVHeaderInfo()
{
    var returnMap = new Dictionary<String, String>();

    FileStream fs = new FileStream(wavFile, FileMode.Open, FileAccess.Read);
    BinaryReader br = new BinaryReader(fs);

    fs.Position = 8;
    returnMap.Add("format", new String(br.ReadChars(4)));

    fs.Position = 22;
    returnMap.Add("channels", br.ReadInt16().ToString());

    fs.Position = 24;
    returnMap.Add("sample rate", br.ReadInt32().ToString());

    fs.Position = 34;
    returnMap.Add("bits per sample", br.ReadInt16().ToString());

    return returnMap;
}
```

Kod 3: Funkcja odpowiadająca za sczytywanie informacji z nagłówka pliku WAV

3.2. Odtwarzanie dźwięku z plików mp3.

Do odtwarzania dźwięku w formacie mp3 wykorzystana została instancja, dostarczanego z systemem Windows, odtwarzacza Windows Media Player (z biblioteki *WMPLib*). Dostarczony z biblioteką interfejs graficzny nie został użyty. Tak jak w przypadkach pliku WAV do odtwarzacza najpierw trzeba dostarczyć ścieżkę do pliku. Następnie wystarczy wywołać odpowiednią metodę klasy *IWMPControls* - w tym przypadku *Play()* i *Stop()*.

3.3. Nagrywanie dźwięku.

Nagrywanie dźwięku zrealizowano za pomocą biblioteki NAudio. Po wciśnięciu przycisku tworzona jest instancja odpowiedzialna za przechwytywanie dźwięku, a następnie zmieniana jest jej konfiguracja. Kolejno do instancji dodawane są nowe *EventHandlery* pozwalające obsłużyć zdarzenia: *DataAvailable* – dostępne są dane do zarejestrowania oraz *RecordingStopped* – zatrzymano proces nagrywania.
Tworzona jest instancja odpowiedzialna za zapis nagrania do pliku – wymaga określenia ścieżki do pliku oraz jego konfiguracji. W końcowym etapie rozpoczynane jest nagrywanie za pomocą wywołania metody *startRecording()* obiektu *recordInstance*. Funkcja odpowiedzialna za zatrzymania nagrywania sprowadza się do wywołania bliźniaczo podobnej metody *stopRecording()*.

Funkcja *recordInstance_DataAvailable* obsługuje zdarzenie związane z dostępnymi danymi, zapisuje zawartość buforu do pliku związanego z instancją *recordWriter* odpowiedzialną za zapis, na koniec czyszczona jest instancja odpowiedzialna za zapis.

Funkcja *recordInstance_RecordingStopped* sprowadza się do pozbycia się instancji odpowiedzialnych za nagranie i zapis wykonywanego nagrania.

```
public void startRecord()
{
    recordInstance = new WaveIn();
    recordInstance.WaveFormat = new NAudio.Wave.WaveFormat(48000, 1);

    recordInstance.DataAvailable += new EventHandler<WaveInEventArgs>(recordInstance_DataAvailable);
    recordInstance.RecordingStopped += new EventHandler<StoppedEventArgs>(recordInstance_RecordingStopped);

    recordWriter = new WaveFileWriter(recordFile, recordInstance.WaveFormat);
    recordInstance.StartRecording();
}

1 reference
public void stopRecord()
{
    recordInstance.StopRecording();
}
```

Kod 4: Główne funkcje odpowiadające za nagrywanie dźwięku z mikrofonu

```
private void recordInstance_DataAvailable(object sender, NAudio.Wave.WaveInEventArgs e)
{
    if (recordWriter == null) return;
    recordWriter.Write(e.Buffer, 0, e.BytesRecorded);
    recordWriter.Flush();
}

1 reference
private void recordInstance_RecordingStopped(object sender, NAudio.Wave.StoppedEventArgs e)
{
    this.recordWriter.Dispose();
    this.recordWriter = null;
    recordInstance.Dispose();
}
```

Kod 5: Funkcje obsługujące zdarzenia nowych danych i rozpoczęcia nagrywania

4. Wnioski

Przygotowana aplikacja poprawnie realizuje wszystkie funkcjonalności założone w ramach laboratorium. Dzięki wykorzystaniu gotowych bibliotek udało nam się w różny sposób wykorzystać możliwości karty dźwiękowej.

Przy realizacji zadań pojawiły się również problemy, spowodowane głównie przez wsparcie bibliotek, które są często przestarzałe. Pojawił się między innymi kłopot z używaniem DirectSounda, gdyż jest to już niewspierana część DirectXa – Visual Studio nie chciało zezwolić na użycie niektórych z jej komponentów. Ostatecznie jednak wyłączenie wykrywania jednego z wyjątków pozwoliło na swobodne używanie aplikacji. Kolejnym problemem związanym z tą biblioteką była zmiana „panningu”. Przy tworzeniu programu pojawiał się wyjątek, który uniemożliwiał zmianę tej wartości. Okazało się jednak, że aby edytować poszczególne wartości odtwarzacza, należy najpierw włączyć możliwość jej edycji przy tworzeniu obiektu klasy *BufferDescription*.