온라인 게임에서의 MIDI사용

(UWP MIDI API소개)

유영천

Tw:@dgtman

https://megayuchi.com

MIDI?

Musical Instrument Digital Interface, 줄여서 MIDI(미디, /ˈmɪdi/))는 전자 악기끼리 디지털 신호를 주고 받기 위해 각 신호를 규칙화한 일종의 규약이다. 다시 말해 악기와 컴퓨터, 악기와 악기끼리 주고받을 수 있는 언어와 통로의 신호 체계 표준이라 할 수 있다. 어떤 전자 악기(건반, 신시사이저, 모듈 등)가 이 표준에 따라 만들어졌다면, 그 전자 악기가 미디를 지원한다고 할 수 있다.

https://ko.wikipedia.org/wiki/MIDI

MIDI 표준 비교

- https://ko.wikipedia.org/wiki/MIDI_%ED%91%9C%EC%A4%80_%EB%B9%84%EA%B5%90

게임에서의 사용(과거)

- 80/90년에는 BGM재생을 위해 사용했다.
  - MT-32 - (80/90년대 PC게이머라면 누구나 꿈꿨지만 사용해본 사람은 극히 드물다)
  - SC-55 Pro, SC-88 Pro – 당시 컴퓨터 음악 한다는 사람들은 물론 주머니 사정이 넉넉한 파워 유저들의 허세템.
- 지금은 거의 모든 게임이 BGM을 wav형태의 파일로 재생한다.
- 일부 건반을 이용하는 게임에서 MIDI 입출력을 사용한다.

게임에서의 사용(현재)

- 자유로운 연주가 필요한 경우
  - 모든 Windows Device(XBOX포함)는 Software Midi synth를 내장하고 있다.
  - 웬만한 OS에서는 MIDI API와 Soft synth를 지원한다. 모바일 OS에서도 소프트 미디지원(https://developer.android.com/ndk/guides/audio/midi)
  - CPU부하가 낮다.
  - 저렴한 마스터 키보드를 손쉽게 입력 장치로 사용 가능

- 온라인 게임에서 연주기능으로 사용할 수 있다.
  - 엄청나게 적은 데이터 사이즈(패킷 사이즈)

Windows에서의 MIDI

- Microsoft GS Wavetable Synth라는 GS모드 호환 소프트웨어 MIDI디바이스를 내장하고 있음.
- 고전적인 데스크탑 API
  - https://docs.microsoft.com/en-us/windows/win32/multimedia/midi-functions

- 새로운 UWP API
  - https://docs.microsoft.com/ko-kr/windows/uwp/audio-video-camera/midi

UWP API

- Windows.Devices.Midi
  - C++/CX , C++/winrt , C#으로 사용 가능
  - Fxxxxxx MS는 C#샘플만 제공함.
  - C#샘플코드를 참고해서 C++/CX로, C++/CX코드를 C++/winrt코드로 포팅 가능.
  - Windows 데스크탑 어플리케이션에서도 사용 가능.
  - Midi API뿐 아니라 모든 UWP API는 따로 DLL로 격리해서 사용할 것!
  - XBOX에서도 사용 가능.
- https://docs.microsoft.com/en-us/uwp/api/windows.devices.midi?view=winrt-20348
- https://docs.microsoft.com/ko-kr/windows/uwp/audio-video-camera/midi

구현

- Windows::Devices::Enumeration::DeviceWatcher 객체를 이용해서 Midi Device를 열거.
- Midi Device로부터 IMidiInPort 객체를 얻어서 입력 메시지 핸들러 세팅
- Midi Deivce로부터 IMidiOutPort 객체를 얻어서 음계를 출력

음계출력

```
BOOL RefMidi::NoteOn(unsigned char channel, unsigned char note, unsigned char Velocity)
{
    BOOL    bResult = FALSE;

    if (!m_SelectedMidiOutDevice)
        return FALSE;

    IMidiOutPort^ pPort = m_SelectedMidiOutDevice->GetPort();

    MidiNoteOnMessage^ MidiNote = ref new MidiNoteOnMessage(channel, note, Velocity);
    pPort->SendMessage(MidiNote);

    return TRUE;
}
BOOL RefMidi::NoteOff(unsigned char channel, unsigned char note, unsigned char Velocity)
{
    BOOL    bResult = FALSE;


    if (!m_SelectedMidiOutDevice)
        return FALSE;

    IMidiOutPort^ pPort = m_SelectedMidiOutDevice->GetPort();

    MidiNoteOffMessage^ MidiNote = ref new MidiNoteOffMessage(channel, note, Velocity);
    pPort->SendMessage(MidiNote);

    return TRUE;
}
```

음계입력

```
void RefMidi::OnAddedMidiInDevice(Windows::Devices::Enumeration::DeviceWatcher^ watcher, Windows::Devices::Enumeration::DeviceInformation^ Info)
{
    wstring wstrDeviceName(Info->Name->Data());
    wstring wstrDeviceID(Info->Id->Data());

    IMidiInPort^ Port = GetMidiInPort(Info->Id);
    if (Port)
    {
        Port->MessageReceived +=
            ref new Windows::Foundation::TypedEventHandler<Windows::Devices::Midi::MidiInPort ^, Windows::Devices::Midi::MidiMessageReceivedEventArgs ^>(this, &RefMidi::OnMessageReceived);
        std::wstring wstrDeviceId(Info->Id->Data());
        MIDI_IN_DEVICE^ Device = ref new MIDI_IN_DEVICE(Port, Info->Name, Info->Id);

        AcquireSRWLockExclusive(&m_Lock_MidiInDeviceCollection);

        auto it = m_pMidiInDeviceList->find(wstrDeviceId);
        if (it == m_pMidiInDeviceList->end())
        {
            WriteDebugStringW(DEBUG_OUTPUT_TYPE_DEBUG_CONSOLE, L"[MidiLib] OnAddedMidiInDevice - %s , %s\n", wstrDeviceName.data(), wstrDeviceID.data());
            m_pMidiInDeviceList->insert(make_pair(wstrDeviceId, Device));
        }
        ReleaseSRWLockExclusive(&m_Lock_MidiInDeviceCollection);
    }
}
```

이벤트 핸들러

음계입력

```
void RefMidi::OnMessageReceived(Windows::Devices::Midi::MidiInPort ^ Port, Windows::Devices::Midi::MidiMessageReceivedEventArgs^ args)
{
    IMidiMessage^ receivedMidiMessage = args->Message;

    receivedMidiMessage->RawData;
    switch (receivedMidiMessage->Type)
    {
        case MidiMessageType::NoteOn:
            {
                MidiNoteOnMessage^ NoteOnMessage = static_cast<MidiNoteOnMessage^>(receivedMidiMessage);
                unsigned char Note = NoteOnMessage->Note;
                unsigned Vel = NoteOnMessage->Velocity;
                unsigned char Channel = NoteOnMessage->Channel;
                m_pOnMidiInputCallback(TRUE, Channel, Note, Vel, m_pCallbackData);
            }
            break;
        case MidiMessageType::NoteOff:
            {
                MidiNoteOffMessage^ NoteOffMessage = static_cast<MidiNoteOffMessage^>(receivedMidiMessage);
                unsigned char Note = NoteOffMessage->Note;
                unsigned Vel = NoteOffMessage->Velocity;
                unsigned char Channel = NoteOffMessage->Channel;
                m_pOnMidiInputCallback(FALSE, Channel, Note, Vel, m_pCallbackData);
            }
            break;
    }
}
```

음계입력

네트워크를 통한 연주

- 인접한 캐릭터들에게 음계 데이터를 브로드캐스팅.
- 인접의 기준은 채팅/이동처리 등을 브로드캐스팅 하는 서버에서의 섹터단위.
- 건반을 누르는 족족 즉시 전송하면 그 자체로 DDOS가 될 수 있음. 약간의 딜레이를 허용하자.
- 어차피 패킷은 뭉쳐서 전송될 수 있음.
- 대신 음계 데이터 사이의 간격이 정확히 유지되어야 하므로 이전 음계 데이터 대해 상대적인 시간값을 패킷에 포함한다.

패킷 구조