オブジェクト指向とゲームプログラミング

▶ DirectX Audio編 ― 第９回　サウンドバッファの再生と停止 ◀

ボイス管理

DirectSoundでは、サウンドバッファの再生時にボイス管理を行うことができます。

ボイス管理とは、サウンドカードのリソースを最大限に利用するために、サウンドバッファへのリソースの割り当てを再生時まで遅延したり、優先順位の低いサウンドを早めに停止してそのリソースを解放し、新しいサウンドバッファにそのリソースを割り当てることをいいます。
通常、サウンドバッファはハードウェアに割り当てられ、バッファが破棄されるまで存在し続けます。ハードウェアに空きがない場合は、ソフトウェア(メインメモリ)に割り当てられ、CPUでバッファの管理とミキシングを行います。ハードウェアに割り当てられる数は環境により異なりますが、多くのバッファを作成する場合、一部のバッファはソフトウェアに割り当てられる可能性があります。
ボイス管理を使うと、バッファの再生時にリソースの割り当てが行えるので、ハードウェアの利用制限を回避することができます。ボイス管理を行うには、セカンダリバッファの生成時にDSBCAPS_LOCDEFERフラグを指定する必要があります。

サウンドバッファの再生

サウンドバッファの再生は、IDirectSoundBuffer8::Playメソッドで行います。再生は、現在の再生カーソル位置から行われます。再生カーソルは、IDirectSoundBuffer8::SetCurrentPositionメソッドで変更することができます。０を指定すると先頭に戻ります。

IDirectSoundBuffer8::Playメソッド

―説明―
Playメソッドは、サウンドバッファを再生カーソルの位置から再生します。

―書式―
```
HRESULT Play(DWORD dwReserved1, DWORD dwPriority, DWORD dwFlags);
```

―パラメータ―
１つ目の引数(dwReserved1)は、予約済なので０でなければなりません。

２つ目の引数(dwPriority)は、サウンドの優先度です。優先度はボイス管理で使用します。最も低い優先度は0、最も高い優先度は0xFFFFFFFFです。バッファがDSBCAPS_LOCDEFERフラグで作成されていない場合、この値は0でなければなりません。

３つ目の引数(dwFlags)は、バッファの再生方法を指定するフラグです。以下のフラグを組み合わせます。

・ループフラグ

DSBPLAY_LOOPING	ループ再生を行います

・ボイス割り当てフラグ
バッファがDSBCAPS_LOCDEFERフラグで作成されている場合に有効です。

DSBPLAY_LOCHARDWARE	ハードウェアで再生します
DSBPLAY_LOCSOFTWARE	ソフトウェアで再生します

・ボイス管理フラグ
バッファがDSBCAPS_LOCDEFERフラグで作成されている場合に有効です。ハードウェアに空きがない場合に新しいバッファ用の空きを確保する方法を指定します。

DSBPLAY_TERMINATEBY_PRIORITY	優先度が最も低いバッファを停止します
DSBPLAY_TERMINATEBY_TIME	再生の残り時間が少ないバッファを停止します
DSBPLAY_TERMINATEBY_DISTANCE	DSBCAPS_MUTE3DATMAXDISTANCEフラグが設定され、最大距離を超えているバッファを停止します。このようなバッファがない場合は失敗します。

ボイス割り当てフラグを指定しない場合は、メソッドが呼び出されたときのリソースの空き状況により、サウンドはソフトウェアまたはハードウェアで再生されます。

－戻り値－
成功した場合はDS_OK、それ以外はエラーの原因をエラーコードで返します。

```
// 再生(pDSBufferは初期化済みのDirectSoundBuffer8オブジェクト)
pDSBuffer->Play(0, 0, DSBPLAY_LOOPING);      // ループ再生
```

サウンドバッファの停止

サウンドバッファの停止は、IDirectSoundBuffer8::Stopメソッドで行います。停止しても、再生カーソルの位置は先頭に戻りません。続けてPlayメソッドを呼び出せば、停止した位置から再生が再開されます。

```
// 停止
pDSBuffer->Stop();
```

課　題

CSoundBufferクラスに、サウンドバッファの再生と停止機能を追加しましょう。

(１)再生はPlay関数、停止はStop関数として作成します。Play関数には、優先度と動作フラグを渡します。これらをふまえると、各関数のプロトタイプは以下のようになります。ISoundBufferクラスの適切な場所に追加しましょう。

```
virtual bool Play(const DWORD inPriority = 0, const DWORD inFlags = 0) = 0;
virtual void Stop() = 0;
```

(２)CSoundBufferクラスに、Play関数とStop関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool Play(const DWORD inPriority, const DWORD inFlags);
virtual void Stop() { m_pDSBuffer->????(); }
```

(３)CSoundBuffer::Play関数の足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          再　生                                                   */
/*****************************************************************************************************/
bool CSoundBuffer::Play(const DWORD inPriority, const DWORD inFlags)
{
    m_pDSBuffer->※ここは各自考えましょう;      // 再生カーソルを先頭に戻す
    if(※ここは各自考えましょう != DS_OK) {
        ::OutputDebugString("*** Error - バッファ再生失敗(CSoundBuffer_Play)\n");
        return false;
    }

    return true;
}
```

CSoundBuffer::Play関数、Stop関数ともに、Null Objectパターンを用いているため、m_pDSBufferがNULLかどうかを調べる必要はありません。

(４)CNullSoundBufferクラスに、「何もしない」Play関数とStop関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool Play(const DWORD inPriority, const DWORD inFlags) { return true; }
virtual void Stop() {}
```

(５)サウンドバッファが正しく動作するか確認します。次のようにプログラムを追加、変更しましょう。

・CTestSceneクラスのメンバに追加

```
ISoundBuffer*   m_pSE;
```

・CTestScene::CTestScene関数に追加

```
m_pSE = DXAudio().CreateBufferFromFile("Wav\\SE.wav");
m_pSE->Play();    // m_pSE->Play(0, 0)と同じ意味
```

・CTestScene::~CTestScene関数に追加

```
DXAudio().ReleaseAllBuffers();
```

複数のサウンドが同時に再生できるかどうかもテストしましょう。また、セグメントとの同時再生もできるかどうかもテストしましょう。