

メモリリークの検出

C++言語を扱う以上、メモリ管理から逃れることができません。

C#やJavaは、ガベージコレクション(GC)という機能があり、不要となったメモリ領域を自動的に解放してくれますので、プログラミングの難易度は下がりましたが、GCにもデメリットがあります。

- ・ メモリ解放中は負荷が高い
→ 30~60FPS内で処理できないため、ゲーム画面が止まってしまったり、ラグの原因になります
- ・ いつ、メモリ解放が行われるか予測ができない
→ 意図しないタイミングで、ゲーム画面が止まる、ラグる。
- ・ 自動的といっても、プロのプログラマーほど優秀ではない
→ 不要なメモリが残ることもある。

C++言語は自分でメモリ管理を行う必要がある、という言い方もされますが、捉え方によっては、自分でメモリ制御を行うことができる、という心強い言語でもあります。

GCのデメリットと比較して、メモリ解放の負荷が一気に高まらないように、少しずつメモリを解放して、30~60FPSを維持することができ、タイミングも自分で決めることができます。

メモリ管理といっても、そんなに難しい話ではなく、

ローカル変数

```
Player p = Player();
```

メンバ変数

```
private:  
    Player p_;
```

実体を宣言時に生成する場合は、不要になったらメモリ解放してくれますし、動的にメモリを確保するのであれば、明示的にメモリを解放すれば良いです。(newしたらdeleteする)

ローカル変数

```
Player* p = new Player();  
~  
delete p;
```

メンバ変数

```
private:  
    Player* p_;  
  
~  
  
void Init(void)  
{  
    p_ = new Player();  
}  
  
void Release(void)  
{  
    delete p_;  
}
```

ただ、メモリは私たちの目で、明確に確認することがなかなかできず、イメージの話になってしまうので、実感が湧かず、とても難しいモノになってしまっていて、理解が進まない、ということがよく起きます。

そこで、簡単に導入できるメモリリークの検出機能を使って、実際に確かめてみましょう。

プログラムのエントリーポイント(始まる関数)がある、main.cpp に以下の記述をしてください。

```

main.cpp
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#include <DxLib.h>
#include "Application.h"

#ifdef _DEBUG
#define new new(_NORMAL_BLOCK, __FILE__, __LINE__)
#endif

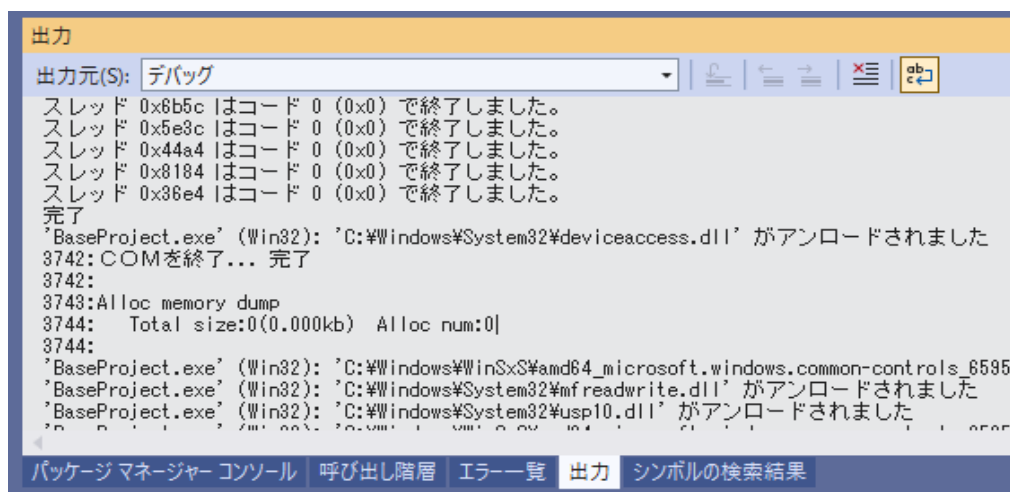
// WinMain関数
//-----
int WINAPI WinMain(
    _In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPSTR lpCmdLine, _In_ int nCmdShow)
{

    // メモリリーク検出
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    ~

```

プログラムを debug モードにして実行しましょう。
しばらくしたら、ゲーム画面を閉じてください。



```

出力
出力元(S): デバッグ
スレッド 0x8b5c はコード 0 (0x0) で終了しました。
スレッド 0x5e8c はコード 0 (0x0) で終了しました。
スレッド 0x44a4 はコード 0 (0x0) で終了しました。
スレッド 0x8184 はコード 0 (0x0) で終了しました。
スレッド 0x36e4 はコード 0 (0x0) で終了しました。
完了
'BaseProject.exe' (Win32): 'C:\Windows\System32\deviceaccess.dll' がアンロードされました
3742: COMを終了... 完了
3742:
3743: Alloc memory dump
3744: Total size:0(0.000kb) Alloc num:0
3744:
'BaseProject.exe' (Win32): 'C:\Windows\WinSxS\amd64_microsoft.windows.common-controls_6595
'BaseProject.exe' (Win32): 'C:\Windows\System32\mfreadwrite.dll' がアンロードされました
'BaseProject.exe' (Win32): 'C:\Windows\System32\usp10.dll' がアンロードされました

```

_CrtSetDbgFlag この関数以降で、メモリ解放し忘れがあったら、出力タブに『memory leaks!』と教えてくれるようになります。
初期状態のプロジェクトでは表示されないかと思います。

では、意図的にメモリリークを起こしてみましょう。

```
// メモリリーク検出
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

int* test = new int();
```

実行して、画面を閉じます。

```
Detected memory leaks!
Dumping objects ->
D:\suekanes\Desktop\PG\2023_02_31\Zlog\0\
[0] C:\Study_02_11\Src\main.cpp(20) :
      {108} normal block at 0x000001C9C61577D0, 4 bytes long.
Data: <    > 00 00 00 00
Object dump complete.
```

メモリリークが発生しました。

int型1つ分になりますので、ちょうど 4bytesの記載があります。

“D:\suekanes\Desktop\PG”～の後が文字化けしていますが、
これは、プロジェクトのパス等で日本語を使用しているのが理由です。

ITでもゲームでも、パスに日本語を使用するのは避けましょう。

次にメモリを明示的に解放してみてください。

```
Detected memory leaks!
```

上記の警告メッセージは無くなったかと思います。

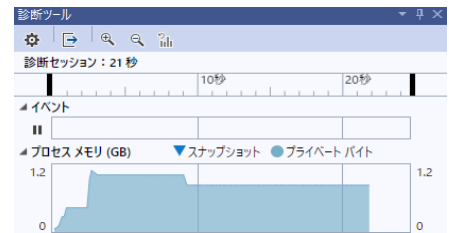
これで、簡単なメモリリークは検出できて無くすことができます。

なぜ、“簡単な”と付け加えたかというと、
プログラム開始時(関数呼ば出し時)と、プログラム終了時のメモリを比較して、
メモリの解放忘れを検出していますので、

例えば、

メモリ使用量

① ゲーム開始	
② タイトルシーン	300M
③ ゲームシーン	500M
④ ゲームオーバーシーン	200M
⑤ タイトルシーン	301M
⑥ ゲームシーン	501M
⑦ ゲームオーバーシーン	201M
⑧ タイトルシーン	302M
⑨ ゲームシーン	502M
⑩ ゲームオーバーシーン	202M
⑪ キチンとメモリ解放	
⑫ ゲーム終了	



このような場合、ゲーム終了時のメモリ解放はキレイにできているため、最後のメモリ検出には引っかからないのですが、明らかにどこかのシーンでメモリリークが発生しています。

この場合、リプレイを何度も何度も繰り返していくと、、、いつかはメモリが不足し、PS5だったら以下のような画面に切り替わるでしょう。



ある程度の目安にはなるかと思いますが、メモリ理解の1つのきっかけにして貰えたらと思います。

もっと良いメモリ検出ツールがあったら教えてください。