```
#ifdef _DEBUG
#include <windows.h>
#include <DxLib.h>
#include "_DebugDispOut.h"
#include "_DebugConOut.h"

std::unique_ptr<_DebugDispOut, _DebugDispOut::_DebugDispOutDeleter> _DebugDispOut::s_Instance(new _DebugDispOut);
_DebugDispOut::_DebugDispOut()
{
    DbgScreen_ = -1;
    waitTime_  = 0;
    _alpha = 255;
    dispFlag_ = true;
    endKey_[0] = 0;
    endKey_[1] = 0;
    pouseKey_[0] = 0;
    pouseKey_[1] = 0;
    homeKey_[0] = 0;
    homeKey_[1] = 0;
    f1Key_[0] = 0;
    f1Key_[1] = 0;
    backSp_[0] = 0;
    backSp_[1] = 0;
    ghBefor_ = 0;
    clsFlag_ = true;
    fpsCount_ = 0;
    fpsView_ = 0;
}

_DebugDispOut::~_DebugDispOut()
{
}

void _DebugDispOut::SetScreen(void)
{
    ghBefor_ = GetDrawScreen();
    SetDrawScreen(DbgScreen_);
    SetDrawBlendMode(DX_BLENDMODE_ALPHA, _alpha);
}

void _DebugDispOut::RevScreen(void)
{
    SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 255);
    SetDrawScreen(ghBefor_);
}

void _DebugDispOut::WaitMode(void)
{
```

```
    f1Key_[1] = f1Key_[0];
    f1Key_[0] = CheckHitKey(KEY_INPUT_F1);
    if (f1Key_[0] && !f1Key_[1])
    {
        TRACE("デバッグ表示ON/OFF          : Home¥n");
        TRACE("デバッグ表示クリアーON/OFF : Delete¥n¥n");

        TRACE("スロー機能(ゆっくり) : PageDown¥n");
        TRACE("スロー機能(は や く) : PageUp¥n");
        TRACE("一時停止/スロー・一時停止解除 : Pause/Break¥n¥n");

        TRACE("バックグラウンド処理有効/無効 : BackSpace¥n");
    }

    if (CheckHitKey(KEY_INPUT_PGDN))
    {
        waitTime_+=10.0;
        TRACE("スロー : %f¥n", waitTime_);
    }
    if (CheckHitKey(KEY_INPUT_PGUP))
    {
        waitTime_ -= 10.0;
        if (waitTime_ < 0.0)
        {
            waitTime_ = 0.0;
        }
        TRACE("スロー : %f¥n", waitTime_);
    }
    pouseKey_[1] = pouseKey_[0];
    pouseKey_[0] = CheckHitKey(KEY_INPUT_PAUSE);

    if (waitTime_)
    {
        startTime_ = std::chrono::system_clock::now();
        do {
            pouseKey_[1] = pouseKey_[0];
            pouseKey_[0] = CheckHitKey(KEY_INPUT_PAUSE);
            if (ProcessMessage() != 0 || CheckHitKey(KEY_INPUT_ESCAPE) == 1)
            {
                break;
            }
            if (pouseKey_[0] && !pouseKey_[1])
            {
                waitTime_ = 0.0;
                TRACE("スロー/一時停止  解除¥n");
                pouseKey_[1] = pouseKey_[0];
            }
            endTime_ = std::chrono::system_clock::now();
```

```
        } while (std::chrono::duration_cast<std::chrono::milliseconds>(endTime_ - startTime_).count() < waitTime_
    }
    if (pouseKey_[0] && !pouseKey_[1])
    {
        TRACE("一時停止¥n");
        waitTime_ = -1.0;
        pouseKey_[1] = pouseKey_[0];
    }

    endKey_[1] = endKey_[0];
    endKey_[0] = CheckHitKey(KEY_INPUT_DELETE);
    if (endKey_[0] && !endKey_[1])
    {
        clsFlag_ ^= 1;
        TRACE("デバッグ表示クリアー機能：%d¥n",clsFlag_);
    }

    backSp_[1] = backSp_[0];
    backSp_[0] = CheckHitKey(KEY_INPUT_BACK);
    if (backSp_[0] && !backSp_[1])
    {
        SetAlwaysRunFlag(GetAlwaysRunFlag() ^ 1);
        TRACE("バックグラウンド処理：%d¥n", GetAlwaysRunFlag());
    }

}

int _DebugDispOut::DrawGraph(int x, int y, int GrHandle, int TransFlag)
{
    SetScreen();
    int rtnFlag = DxLib::DrawGraph(x , y , GrHandle, TransFlag);
    RevScreen();
    return rtnFlag;
}

int _DebugDispOut::DrawBox(int x1, int y1, int x2, int y2, unsigned int Color, int FillFlag)
{
    SetScreen();
    int rtnFlag = DxLib::DrawBox(x1 , y1 , x2 , y2 , Color, FillFlag);
    RevScreen();
    return rtnFlag;
}

int _DebugDispOut::DrawString(int x, int y, char* String, unsigned int Color)
{
    SetScreen();
    int rtnFlag = DxLib::DrawString(x, y, String, Color);
    RevScreen();
```

```
    return rtnFlag;
}
//
//int _DebugDispOut::DrawFormatString(int x, int y, unsigned int Color, std::string FormatString, ...)
//{
//    va_list arglist;
//    va_start(arglist, FormatString);
//    SetScreen();
//    int rtnFlag = DxLib::DrawFormatString(x , y , Color, FormatString.c_str(), va_arg(arglist, int), va_arg(arg
//    RevScreen();
//    va_end(arglist);
//    return rtnFlag;
//}

int _DebugDispOut::DrawLine(int x1, int y1, int x2, int y2, unsigned int Color)
{
    SetScreen();
    int rtnFlag = DxLib::DrawLine(x1 , y1 , x2 , y2 , Color);
    RevScreen();
    return rtnFlag;
}

int _DebugDispOut::DrawCircle(int x, int y, int r, unsigned int Color, int FillFlag)
{
    SetScreen();
    int rtnFlag = DxLib::DrawCircle(x , y , r, Color, FillFlag);
    RevScreen();
    return rtnFlag;
}

int _DebugDispOut::DrawPixel(int x, int y, unsigned int Color)
{
    SetScreen();
    int rtnFlag = DxLib::DrawPixel(x , y , Color);
    RevScreen();
    return rtnFlag;
}

#define FPS_BOX_SIZE_X 80
#define FPS_BOX_SIZE_Y 24
void _DebugDispOut::DrawFPS(void)
{
    fpsEndTime_ = std::chrono::system_clock::now();
    if (std::chrono::duration_cast<std::chrono::milliseconds>(fpsEndTime_ - fpsStartTime_).count() >= 1000)
    {
        fpsView_ = fpsCount_;
        fpsCount_ = 0;
        fpsStartTime_ = fpsEndTime_;
```

```
    }
    else
    {
        fpsCount_++;
    }
    _DebugDispOut::DrawBox(fpsPosX, fpsPosY, fpsPosX + FPS_BOX_SIZE_X, fpsPosY + FPS_BOX_SIZE_Y, 0, true);
    _dbgDrawFormatString(fpsPosX + 4, fpsPosY + 4, 0xffffff, "fps:1/%d", fpsView_);
}

void _DebugDispOut::SetDrawPosFps(FPS_SIDE side, FPS_VER ver)
{
    if (side == FPS_SIDE::LEFT)
    {
        fpsPosX = 0;
    }
    else
    {
        fpsPosX = screenSizeX_- FPS_BOX_SIZE_X;
    }
    if (ver == FPS_VER::TOP)
    {
        fpsPosY = 0;
    }
    else
    {
        fpsPosY = screenSizeY_ - FPS_BOX_SIZE_Y;
    }
}

bool _DebugDispOut::StartDrawDebug(void)
{
    int ghBefor;
    ghBefor = GetDrawScreen();
    SetDrawScreen(DbgScreen_);
    if (clsFlag_)
    {
        ClsDrawScreen();
    }
    SetDrawScreen(ghBefor);
    return true;
}

bool _DebugDispOut::AddDrawDebug(void)
{
    homeKey_[1] = homeKey_[0];
    homeKey_[0] = CheckHitKey(KEY_INPUT_HOME);
    if (homeKey_[0] && !homeKey_[1])
    {
```

```
        TRACE("デバッグ表示：%d¥n", dispFlag_);
        dispFlag_ ^= 1;
    }
    if (dispFlag_)
    {
        SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 255);
        DxLib::DrawGraph(0, 0, DbgScreen_, true);
        //lpSceneMng.AddDrawQue({ _DbgScreen,lpSceneMng.ScreenSize.x/2,lpSceneMng.ScreenSize.y / 2,0,INT_MAX,LAYE
    }
    WaitMode();
    return true;
}

bool _DebugDispOut::SetAlpha(int alpha)
{
    _alpha = alpha;

    return true;
}

bool _DebugDispOut::Setup(int screenSizeX, int screenSizeY,int alpha)
{
    if (DbgScreen_ == -1)
    {
        DbgScreen_ = MakeScreen(screenSizeX, screenSizeY, true);
    }
    SetAlpha(alpha);
    screenSizeX_ = screenSizeX;
    screenSizeY_ = screenSizeY;

    return true;
}

bool _DebugDispOut::SetWait(double timeCnt)
{
    waitTime_ = timeCnt;
    return true;
}

#endif    // _DEBUG
```