DxLib の導入と使い方の初歩

TEA

DX ライブラリとは

DirectX の機能を簡単に扱えるようにしたライブラリで、最近では 2D だけではなく樋口 M 氏が開発した MikuMikuDance で使われているモデルデータ(.pmd)を読み込んで扱えるなど 3D も扱えるようになっています。基本的に公式サイトで十分ですのでネット環境がある場合で困ったら公式サイトを、オフライン環境の場合は DX ライブラリの中にある help を見ましょう。(両者の内容は同じです。)

DxLib の導入について

・VS の設定をするまえに……

1.公式から DxLib をダウンロード。

(URL: http://homepage2.nifty.com/natupaji/DxLib/)

このページの真ん中辺りにある DX ライブラリのダウンロードのページの VisualC++用をダウンロードします。

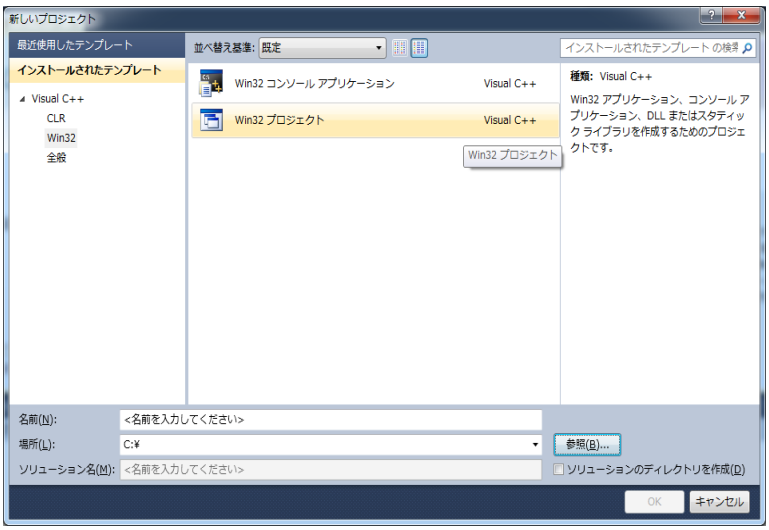
2.ダウンロードしてきた DxLib__VCx_xxx.EXE を実行(x_xxx にはバージョンが入ります。)。

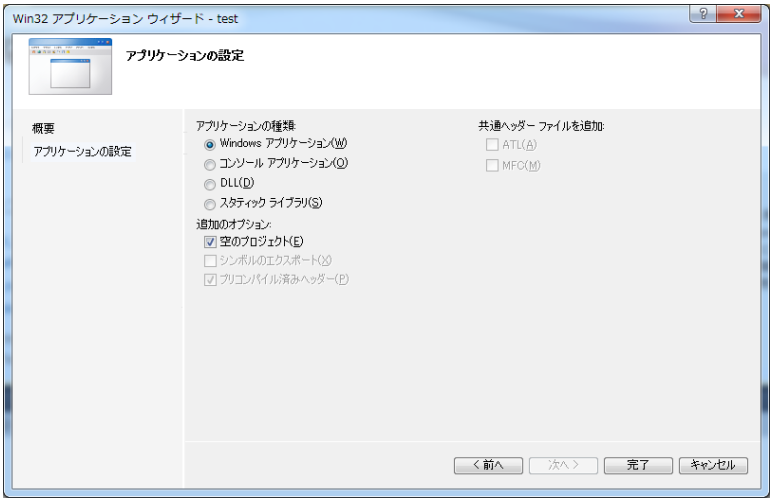
3.できたフォルダ(名前は DxLib_VC)を絶対に動かさないような所に移動させて下さい(私の場合は C ドライブ直下に置いてます。)。

・ここから VS の設定に関することを

1.新しいプロジェクトを Win32 アプリケーション(※コンソールではない)で作成。名前は適当に。空のプロジェクトのチェックボックスにチェックを入れることを忘れずに。

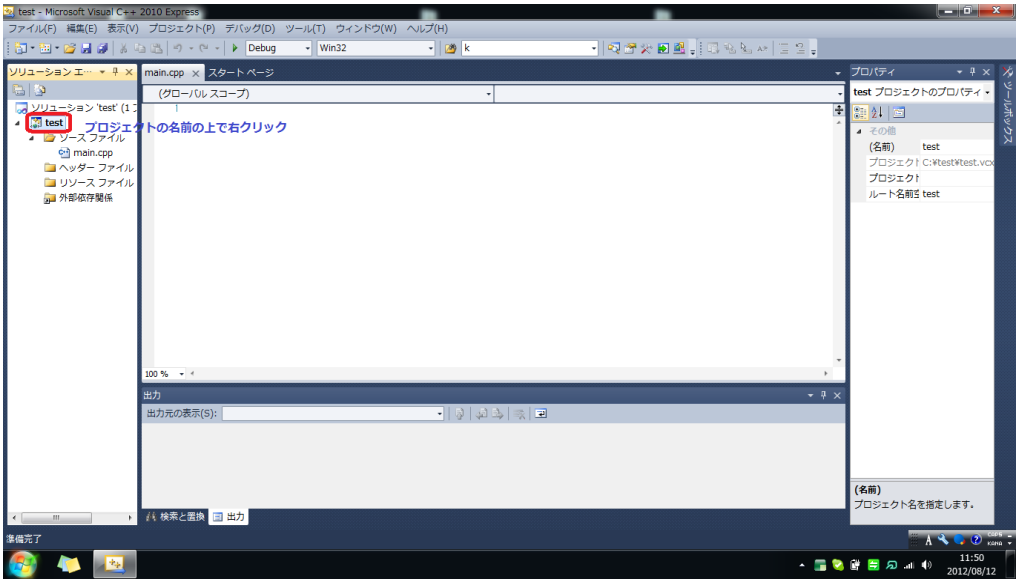
(以下の画像は私、TEA の PC の環境で撮影したスクリーンショットであり、人により多少の誤差が出るかも知れません。スクリーンショットの撮影に使ったものは VisualC++2010Express です。左記と異なるバージョンを使っている人は多少画像と違う点があることをご了承して御覧ください。)



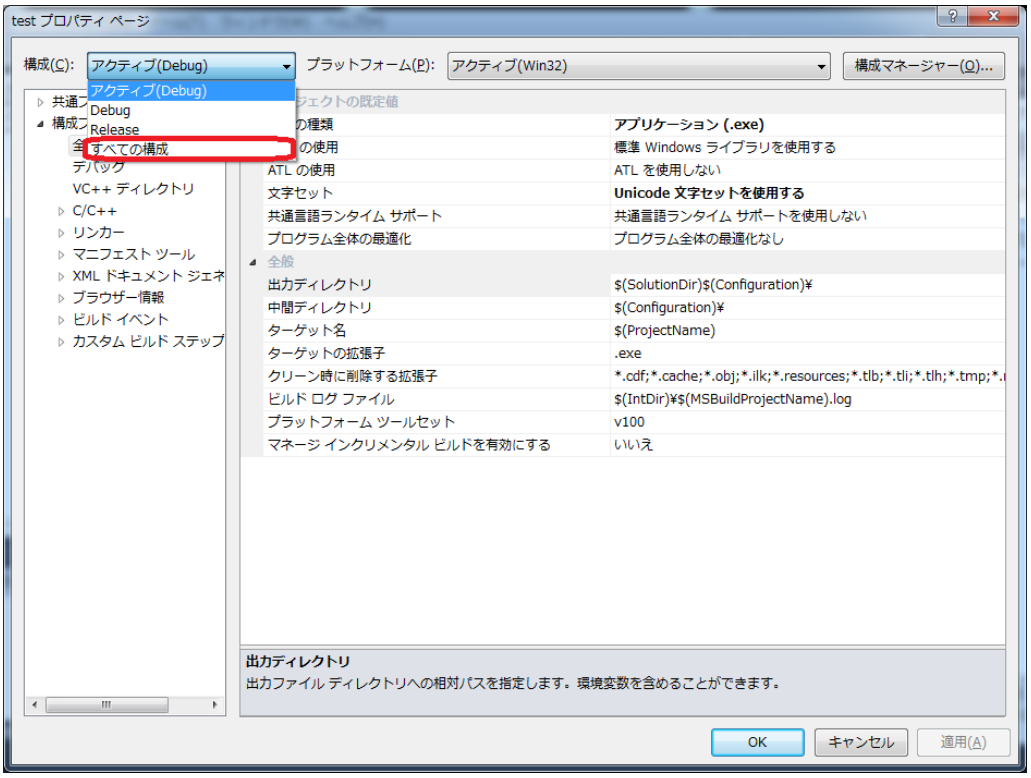


2.ソースファイルの cpp ファイルを追加。

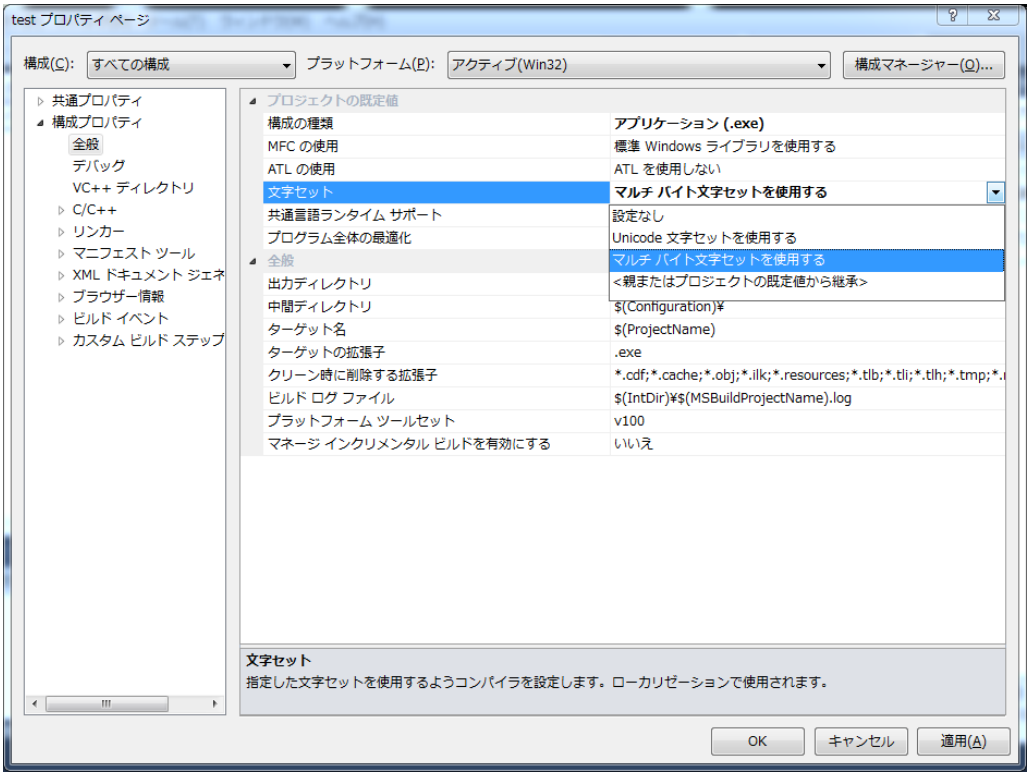
3.画像の所で右クリックしプロパティを選択。



4.すべての構成に変更

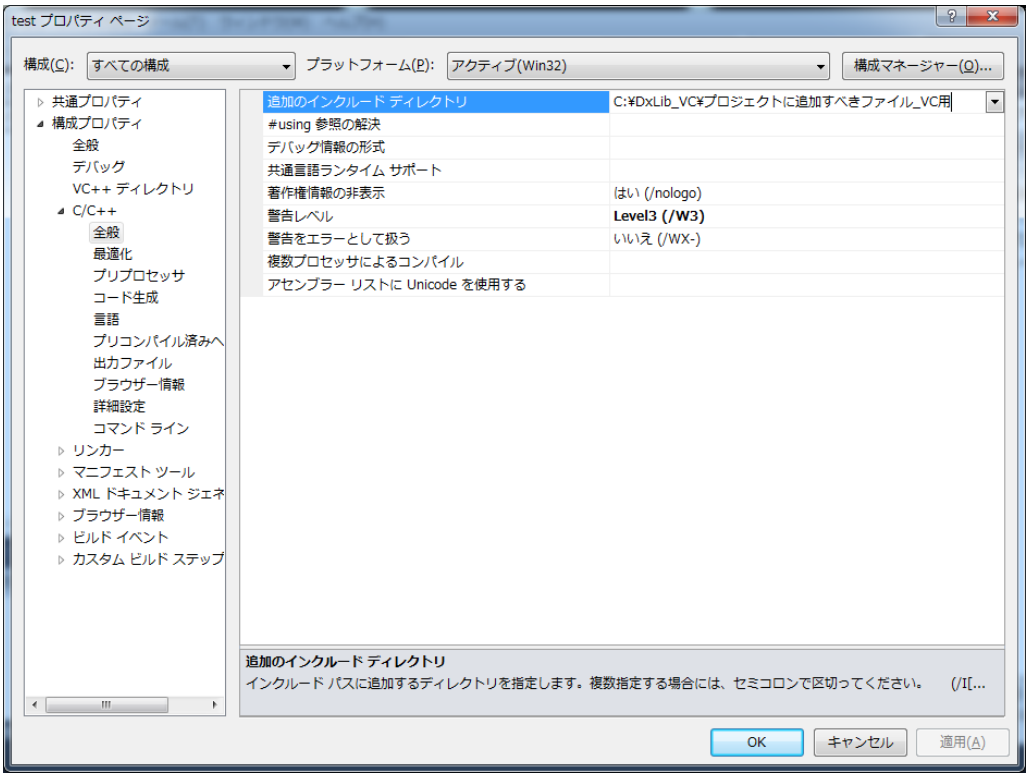


5.Unicode 文字セットを使用するをマルチバイト文字セットを使用するに変更。

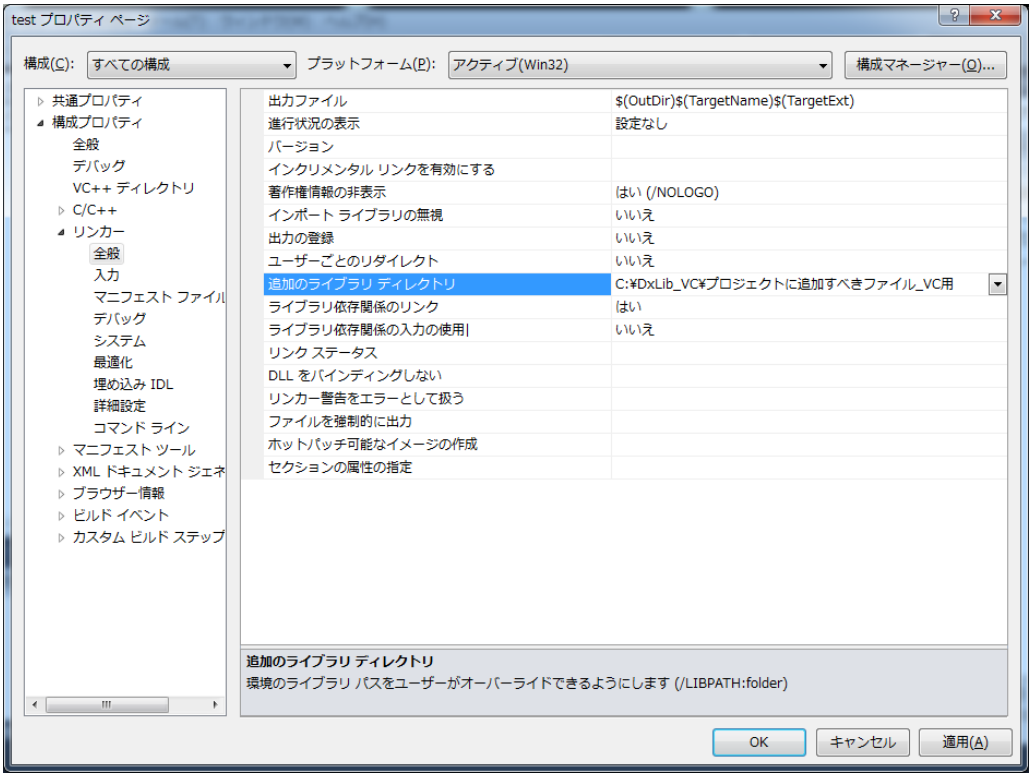


6.C/C++の全般の追加のインクルードディレクトリを選択。

7.先ほど移動させた DxLib_VC 内のプロジェクトに追加すべきファイル_VC 用のフォルダのパスを指定。

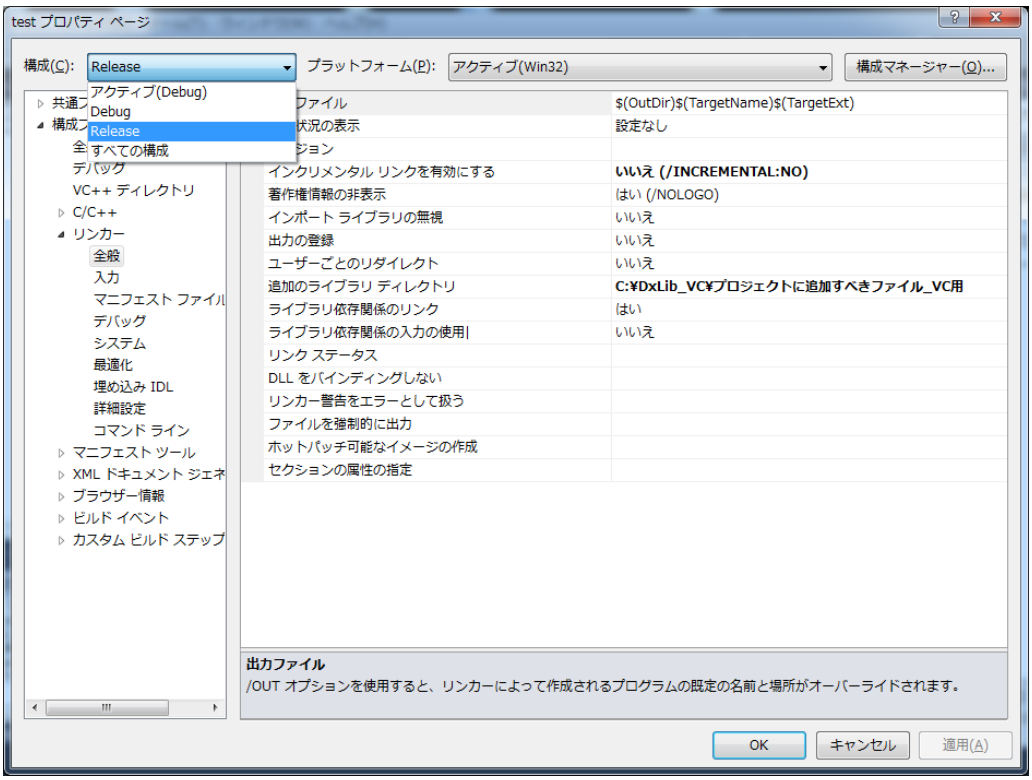


8.リンカーの全般の追加のライブラリディレクトリにも同じように指定する。



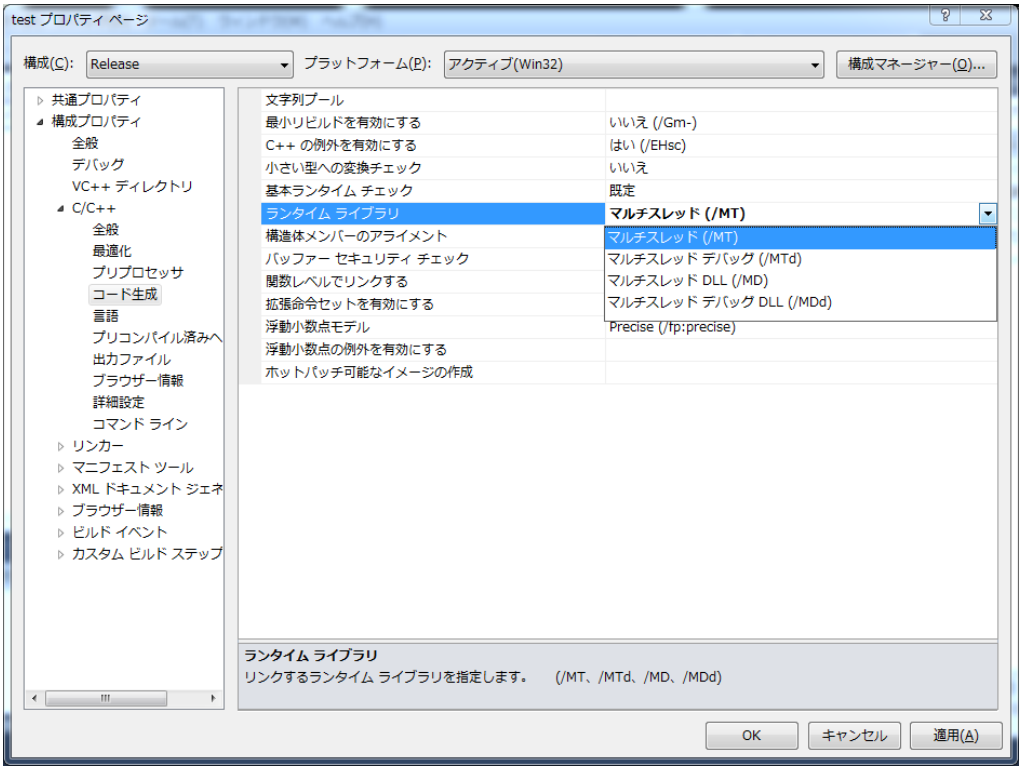
9.右下の適用をクリック。

10.すべての構成から Release に変更。



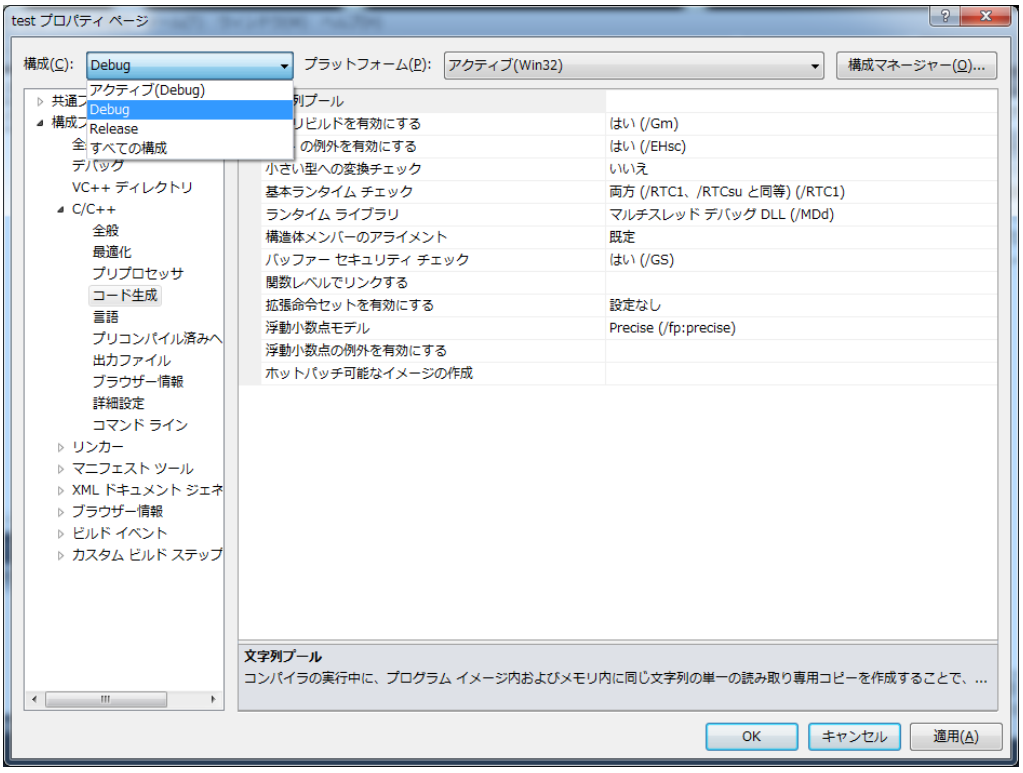
11.C++のコード生成を選択。

12.ランタイムライブラリの項目をマルチスレッド(/MT)に変更。

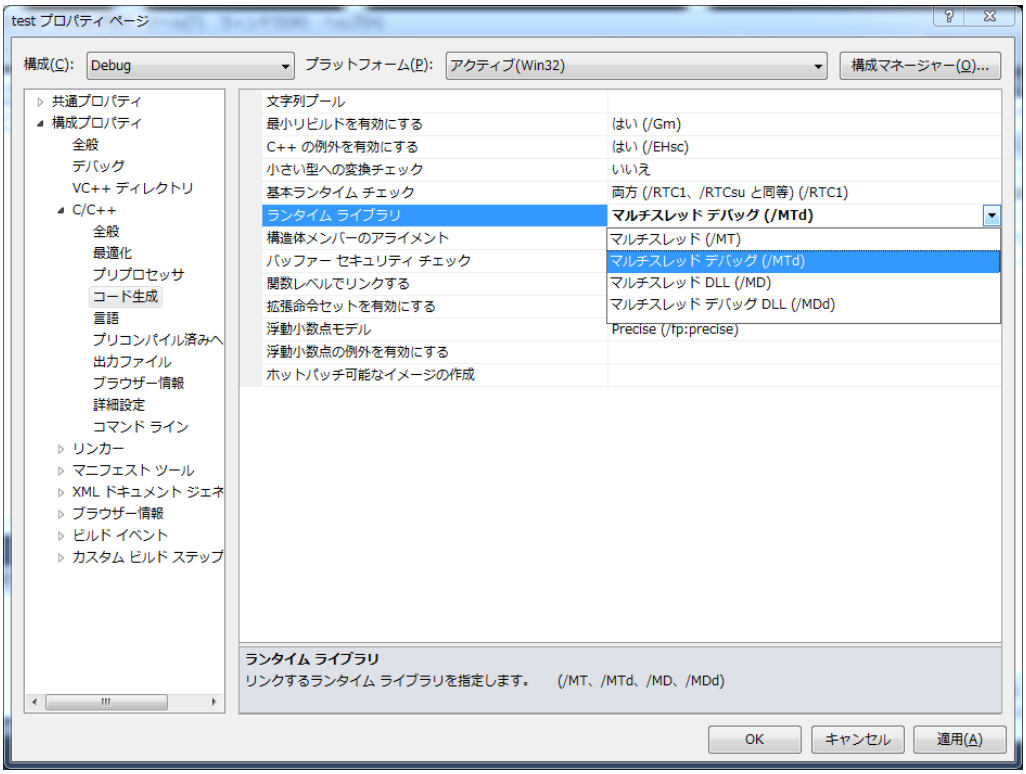


13.右下の適用をクリック。

14.Release を Debug に。



15.ランタイムライブラリの項目をマルチスレッドデバッグ(/MTd)に変更。



16.右下の OK をクリックして設定は終了です。お疲れ様でした。

Dxlib を使ったプログラミングについての作法

DxLib を使う際は今までとは違って#include”Dxlib.h”のようにインクルードして下さい。“”を使う方は標準のライブラリにはない自分で制作したヘッダーファイルなどをインクルードする際に使うものです。注意して下さい。

今までやってきた C 言語もしくは C++ではメイン関数を
int main()
のように書いてきましたが Dxlib をつかったプログラムはそのようには書きません。
Dxlib をつかったプログラミングは Windows プログラミングに分類され、Windows プログラミングにはそれ独特の作法や規則があるからです。そのなかでも Dxlib をつかったプログラミングで絶対に使うのがメイン関数の部分で、Windows プログラミングではメイン関数を
int WINAPI WinMain(HINSTANSE hInstanse , HINSTANSE hPrevInstanse , LPSTR lpCmdLine , int CmdShow)
と書きます。この引数や見たこと無い HINSTANSE などは C 言語や C++で使っている int 等と同じで Windows プログラミングで使う型名の一つですが覚えなくて結構ですし、これはコピペで使って下さい。覚えずにコピペで使って下さい。

また、DX ライブラリの使用する際は初期化用の関数を呼び出す事を忘れないで下さい。初期化用の関数は Dxlib_Init という関数で何らかのエラーが起きたときは返り値が－1 です。また Dxlib_Init 関数を呼び出した際は必ず Dxlib_End 関数を呼び出して下さい。
　ここまでで書いたことをまとめると以下のコードが DxLib を使用した場合の最少のコードになります。

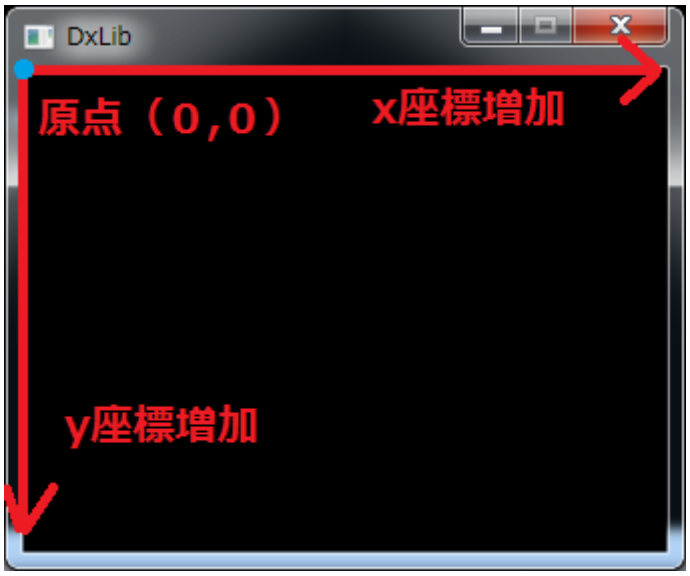
```
#include"Dxlib.h"
int WINAPI WinMain(HINSTANCE hINnstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine, int CmdShow)
{
	if(DxLib_Init() == -1)
	{
		return -1;
	}

	DxLib_End();
	return 0;
}
```

ゲームを作る上で必要な概念とその実装の仕方

・画面上の座標系について

コンソールの時とは違い Dxlib を使ったプログラミング(というか DirectX を使ったプログラミング)では座標の概念が加わります。DxLib では左手系と言われる座標系を採用しており左右が X 軸、上下が Y 軸、奥行きが Z 軸で X 軸は右向きを正、Y 軸は下向きを正、Z 軸は奥に行くほうが正となります。多くの人がやるであろう 2D の場合、原点は左上で下に行くほど y 座標は増加、右に行くほど x 座標は増加します。具体的にどのようになっているかというと次の画像を見て理解して下さい。



・fps(frames per second/フレームレート)について

> **フレームレート**は、動画において、単位時間あたりいくつフレーム(コマのこと)が処理されるか、という値である。通常、1 秒あたりの数値で表し、fps(Frames Per Second、フレームス パー セコンド)という単位で表す。
> (引用元:Wikipedia の“フレームレート”の項目より)

通常のゲームでは 60fps(一秒間に 60 枚のコマが出ている)で動いており、30fps だとややカクカクした状態、15fps を切ってしまうとかなりカクカクした状態になってしまうので気をつけて下さい。また、処理落ちがかかって 60fps のゲームが 30fps になった場合は 1 コマあたりにかかる時間が 2 倍になっておりゲームの速度が 1/2 になっているということを言っています。

60fps の場合 1 フレームの処理は 1(s)=1000(ms)なので 1000(ms)/60(fps)=16.6666(ms)になり約 0.016 秒になります。

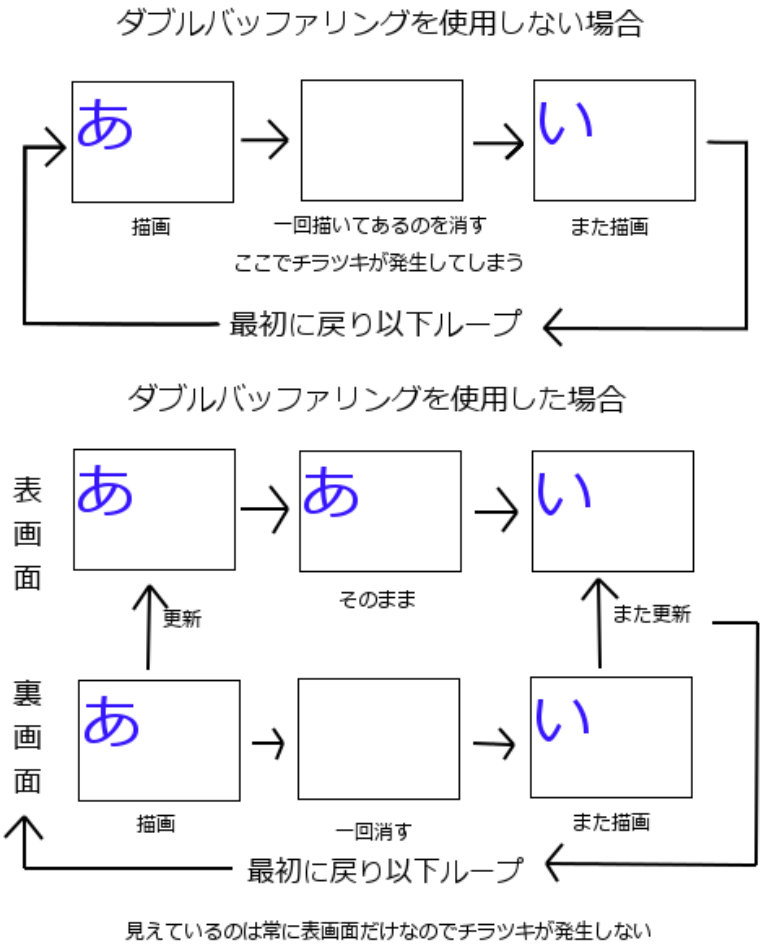
調節の仕方はメインループ(後述)の始まりの部分で時間を取得、ループの終わりで時間を取得しその差が 16.6666(ms)になるまで止めるなどの方法があります。

・メインループについて

ゲームでは 1 フレームごとに 1 ループさせ、そのループの間に敵の位置を移動させたり、表示させたりなどの処理を行います。このループはゲームを止めるまでずっとこの中にいるため無限ループ(厳密に言うとエラーが発生するまで無限ループ)にしておく必要がありこのループをメインループと呼びます。

・ダブルバッファリングについて

簡単に言うとダブルバッファリングは画面更新に伴う「ちらつき」を防ぐための技術です。画面に表示するもの(表画面)と画面には表示しないもの(裏画面)2 つの画面をまず用意します。そして、裏画面に全て描画した後にそれを表画面にコピー、裏画面の絵を一回消して再描画、表画面へコピー、……を繰り返すことにより一回消すときにどうしても発生してしまうチラツキを抑えます。具体的には以下のとおりです。



一回描いてある物を消さない場合、上の例だと「あ」と「い」が重なってしまうため消す必要があります。裏画面に描画するという指定をする場合は SetDrawScreen という関数を使い、表画面に裏画面を反映させるには ScreenFlip という関数を、画面を一回まっさらにするには ClearDrawScreen という関数を使います。

・おおまかな関数の概要

1. グラフィック関連とサウンド関連

DX ライブラリでは画像や音を扱う際に「ハンドル」で管理します。ハンドルって？と戸惑うことはなくハンドルは単に int 型で宣言した変数のことです。

一枚絵の画像の場合は int 型で宣言したハンドル(ここでは仮に GHandle としておきましょう)に LoadGraph という関数で読み込ませるだけです。画面に描画する際は主に DrawGraph という関数で描画します。実際の流れは以下のようになります。

```
int GHandle;//ハンドルの宣言
GHandle = LoadGraph("読み込む画像の相対パス");//画像の読み込み
DrawGraph(x座標,y座標,GHandle,透過色を有効にするか否かのフラグ);//画像の描画
```

また、アニメーション用の素材を 1 枚にまとめた画像の場合にはコマ数が要素数と同じようになるような int 型の配列でハンドルを宣言します(ここではコマ数が 16 で配列の名前を GHandle とします。)。読み込みの際は LoadGraph ではなく LoadDivGraph という関数を用います。ここで注意すべきなのは要素の番号がどのように付けられているかです。以下のように番号は付けられます。

例:4*4 に分割する場合

[0]	[1]	[2]	[3]
[4]	[5]	[6]	[7]
[8]	[9]	[10]	[11]
[12]	[13]	[14]	[15]

描画する際は一枚絵と同じように主に DrawGraph という関数を使いますが一定フレームごとに描画する画像のハンドルを変えるということでアニメーションを実現させます。実際の流れは以下のようになります

```
int GHandle[16];//ハンドルの宣言
LoadDivGraph("読み込む画像の相対パス",コマ数,横の分割数,縦の分割数,分割後の横の大きさ,分割後の縦の大きさ,GHandle);//画像の読み込み
DrawGraph(x座標,y座標,GHandle[i],透過色を有効にするか否かのフラグ);//画像の描画
//i は描画する分割後の画像の番号
```

また、複数の大きさが均一でない素材を一枚の画像に収めた時の読み込みの場合には DerivationGraph という関数を使います。この関数は一回読み込んだ画像の左上の座標と抜き出したい画像の大きさを指定して、元の画像から抜き出して新しいグラフィックのハンドルを作成するものです。実際の流れは以下のようになります。

```
int GHandle1,GHandle2;
GHandle1 = LoadGraph("読み込む画像の相対パス");//元画像の読み込み
GHandle2 = DerivationGraph(抜きだしたい部分の左上の点のx座標,
                           抜き出したい部分の左上の点のy座標,
                           抜き出したい部分の幅,
                           抜き出したい部分の高さ,
                           GHandle1);//抜き出しての読み込み
DrawGraph(x座標,y座標,GHandle2,透過色を有効にするか否かのフラグ);//画像の描画
```

以上で画像に関する初歩的な話は終わりです。これだけ使えればかなりのことはできますが他にも描画するための関数として DrawRotaGraph やその拡張版の DrawRotaGraph2、DrawModiGraph などがあります。

さてここからは音に関する話です。音を扱う際もハンドルで管理するということは上のほうで述べた通りで、再生させる方法は描画の方法よりもよく使うものは少ないです。

まず、int 型のハンドルを宣言します(ここでは SHandle とします。)。読み込みの際は LoadSoundMem という関数を使います。そして、再生させる際には PlaySoundMem という関数を使います。実際の流れは以下のようになります。

```
int SHandle;//ハンドルの宣言
SHandle =LoadSoundMem("読み込む音の相対パス");//音の読み込み
PlaySoundMem(SHandle,DX_PLAYTYPE_BACK);//音の再生
```

いきなりここで出てくる DX_PLAYTYPE_BACK というものに疑問を持つかも知れません。これは PLAYTYPE と言う通りにどのように再生させるかを指定するものです。再生方法は 3 種類あるのですが実際に使うのはその内 2 つです。

DX_PLAYTYPE_BACK の場合は一回再生すると止まります。効果音などはこちらを使います。

もう一方の DX_PLAYTYPE_LOOP はループ再生を行います。ゲーム中で使う BGM はこちらのほうがあっているでしょう。用途に応じて使い分けて下さい。

2. 文字関連

コンソールの時と違い Dxlib を使う場合はどこに（x 座標、y 座標）何色で描画するかということも指定してあげなければなりません。コンソールの printf 関数に相当するのは DrawFormatString という関数で実際に使う場合は以下のように書きます。

```
DrawFormatString(x座標,y座標,色の指定,"描画する文字列");//文字列の描画
```

色の指定は GetColor という関数を使います。上には書いてませんが何かしらの変数の値を表示することもでき、その方法は printf 関数と同じです。
この他にも CreateFontToHandle などの様々な関数があります。

3. 入力関連

キーボードからの入力を受け取る際には CheckHitKey という関数をつかいます。実際に使う際は以下のように書きます。

```
if(CheckHitKey(KEY_INPUT_Z)==1)
{
	//キーボードのZキーが押された場合の処理
}
else
{
	//キーボードのZキーが押されていない場合の処理
}
```

DxLib 内で KEY_INPUT_XXXX の形ですべてのキーに番号が振り分けられておりそれを上のように書くことによりキーが押されているかどうかが判定できます。
また複数のキーの入力を認識させたい場合は GetHitKeyStateAll という関数を使います。こちらを使う際は以下のように書きます。

```
char KeyBuf[ 256 ] ;
GetHitKeyStateAll( KeyBuf ) ;

if( KeyBuf[ KEY_INPUT_Z ] == 1 &&
	KeyBuf[ KEY_INPUT_X ] == 1 )
{
	//キーボードのZキーとXキーが押されている場合の処理
}
```

当然のこと(?)なのですが実は、CheckHitKey 関数を使うことにより同じ動作を実現することができます。また、JoyPad からの入力を受け取る GetJoypadInputState という関数やマウスカーソルの位置を受け取る GetMousePoint という関数もあります

・リファレンスについて

Dxlib には多くの関数がありますがすべての関数においてその仮引数の意味、関数名、返り値がどういう意味を持つのか、関数がどのような働きをしているかを覚える必要はありません。ではプログラマはどうやってコードを組むかというとリファレンスを見ながら組みます。リファレンスにはほとんどの関数についての情報が載っています。「○○したいからそれに合う関数が無いかなー？」とおもたらまずリファレンスを見ましょう。そこのなくてもそこに載っている関数を組み合わせることにより自分のやりたいことを実現できる

はずです。ではリファレンスはどこにあるかというとこの資料の一番上で述べた通り DxLib_VC というフォルダ内の help というフォルダに入っている index.html というファイルから飛べます。勿論公式サイトにもあるのでオンライン環境かオフライン環境かでうまく使い分けて下さい。

この資料の最初の方に述べた DX ライブラリの導入についてもリファレンス内にページがあるため、この資料をなくして導入の仕方が分からなくなってしまったという時にも問題なく導入できるはずです。

・隠し関数について

一個前の項で述べましたが「リファレンスにはほとんどの関数についての情報が載っています」という言葉通り殆どの関数が載っています。つまり、言い返せば「すべての関数について載っている」訳ではありません。ここではリファレンスにはのってはいないが Dxlib 内に存在する関数を隠し関数と呼ぶ事にします。なぜ隠しているかはライブラリの制作者様が言うには理由があって隠しているそうです。隠し関数は Dxlib.h 内でプロトタイプ宣言がされているので自由に使うことができますし、コメントにどういった関数なのかのかなり大雑把な説明が書かれているので Dxlib の使用になれた人ならば仮引数名などから何を引数に取りどういった働きをするのかを関数名から推測できるようになっているはずです。

隠し関数を見る場合は、Dxlib.hを読むのはVSで#include"Dxlib.h" の部分をドラッグし右クリック、開いたメニューから「ドキュメント“Dxlib.h”を開く」から読むことができます。また、「DXライブラリ　隠し関数」などのキーワードで検索をかけると様々なページが出てきます。リファレンスに載っている関数では間に合わなくなった際は隠し関数を探してみて下さい。