DXライブラリでのゲームパッド対応について

とりあえず書いておきます。イメージとしてはゼミＢの続きということで。

ゲームパッドというものの意味から。

ゲームパッドは用はゲームのコントローラです。普通に思い浮かべるスーファミやプレステのコントローラのアレです。それ以外にもフライトシミュレーション用のゲームのジョイスティックや USB 対応のアーケードコントローラも入ります。

それで DX ライブラリではそのゲームパッドに対応するために関数があります。

int GetJoypadInputState(int InputType);

という関数です。これは引数の InputType で指定されたパッドからの入力状況を得る関数です。具体的にどう使うかというと

```
if( GetJoypadInputState( DX_INPUT_PAD1) & 1){
    //下ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 2){
    //左ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 4){
    //右ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 8){
    //上ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 16){
    //1 ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 32){
    //2 ボタンが押されている
}
if( GetJoypadInputState( DX_INPUT_PAD1) & 64){
    //3 ボタンが押されている
}
```

以下、4 ボタン、5 ボタンと続いていきます。中身の解説ですが、GetJoypadInputState という関数はパッドの全てのボタンの状態を取得する関数です。よって、１ボタンの入力状態が欲しくても、他の十字キーやボタンの入力状態まで一緒に手に入ってしまいます。

なので、if 文の中でビットアンドというものをしています。説明は面倒なので省きますが、&記号の左側に GetJoypadInputState(DX_INPUT_PAD1)と書いて、右側に２の累乗の数を書いてください。2^0 ～ 2^3 は十字キー、2^4 ～はゲームパッドの１ボタン、２ボタン……となります。32 ビットなので 2^{31} が最大になります。

で、これだけだとリファレンスと変わりませんね。なので実例。
ゼミ B で void m_key(void);という関数を作りましたが、それに組み込んで見ましょう。

```
void m_key( void){

    if( (CheckHitKey( KEY_INPUT_DOWN) ==  1) || (GetJoypadInputState( DX_INPUT_PAD1) & 1)){
        key_up++;
    }
    else{
        key_up = 0;
    }
    if( (CheckHitKey( KEY_INPUT_LEFT) == 1) || (GetJoypadInputState( DX_INPUT_PAD1) & 2)){
        key_left++;
    }
    else {
        key_left = 0;
    }
    //以下略
}
```

こんな感じ。縦棒二つは普通の OR です。つまり、「キーボードの矢印キーが押されている、もしくはパッドの十字キーが押されていれば」、という意味です。こんな感じで他のボタンも対応できるかと思います。

で。
キーコンフィグに関してですが、結構面倒です。
キーコンフィグというのはプレイヤーが自由にキーの場所を変えられることを指します。PC ではゲーム機と違い様々なゲームパッドが市販されているので、ボタンの配置を固定してしまうとパッドによっては非常にやりづらい設計になってしまいます。
代表的な例としてプレイステーションを USB で接続させたときです。一般的なゲームパッドはスーファミでいう B ボタンや A ボタンの位置に 1 番目のボタンが来ることが多い気がします。しかしプレイステーションのパッドは△キーの位置が 1 番目のボタンです。この仕様は本当にやめて欲しいですが、△ボタンがジャンプボタンとか嫌ですね。なのでキーコンフィグを作りましょう。

先ほどの GetJoypadInputState 関数を思い出してください。GetJoypadInputState 関数は全てのボタンの入力状況を受け付ける関数でした。なので&マークが必要なのでした。
しかし逆に考えてください。&マークの右側の数字は別に変数でも構いません。そしてその右側の数字こそが、実際に入力したいキーの番号なのです。なので各キーごとにパッドの番号を記憶しておく変数を用意し、その変数と GetJoypadInputState と&記号で結べば解決します。

具体的にプログラムを書いてみます。

```
void m_key( void){

    //十字キーはキーコンフィグする必要がないので省略
    //
    if( (CheckHitKey( KEY_INPUT_Z) == 1) || (GetJoypadInputState( DX_INPUT_PAD1) & pad_b))
{
        key_a++;
    }
    else{
        key_a = 0;
    }
    if( (CheckHitKey( KEY_INPUT_LEFT) ==  1) || (GetJoypadInputState( DX_INPUT_PAD1) &
pad_b)){
        key_b++;
    }
    else {
        key_b = 0;
    }
    //以下略
}
```

まず、グローバル変数を追加します。int pad_a、int pad_b ……とか使うボタン数の応じてです。ゼミ B で key_a とか変数を宣言しましたが、そこと同じところに書いておけばおｋです。そして m_format 関数あたりに pad_a = 16, pad_b = 32 とか代入してください。これでとりあえずパッド入力ができるはずです。

ではテキストファイルからパッドの情報を読み込む形にしてみましょう。key_config.txt というファイルがあったとします。そして下記のようなファイルの中身だったとします。

```
32 128 64 256
```

一行しかない上に、数字の羅列で意味不明ですが、32 と 128、128 と 64、64 と 256 の間にそれぞれスペースが入っていて、４つ分のボタンの設定という設定にしておきます。これを読み込む関数、get_key_config を書きます。

```
void get_keyconfig_from_text( void){
    FILE *fp;
    fopen_s( &fp, "key_config.txt", "r");
    fscanf_s( fp, "%d %d %d %d", &pad_a, &pad_b, &pad_start, &pad_select);
    fclose( fp);
}
```

あれ、今日の情報処理の授業で似たようなことをやったような……

一応補足すると fscanf_s というのは scanf のテキストファイルからの入力版。テキストから入力された値を pad_a とか pad_b という変数に読み込むという関数。この関数はゲームが起動時に一度だけ実行されればいいということになる。

ここまでくれば、ゲームを起動する前にメモ帳等、テキストエディタで数値を変えることによってキーコンフィグができます。東方萃夢想なんかはゲームではなくキーコンフィグをするソフトがありましたが、あれはこのテキストファイルを作るソフト、と考えればいいのです。別にゲームのなかでキーコンフィグする必要なんてない。

ちょっとそろそろ疲れたのでこれくらにしておきます。ゲーム画面内でのキーコンフィグについては自分で考えてください。