2006 年度 明治大学エレクトロニクス研究部
ソフトゼミ B テキスト

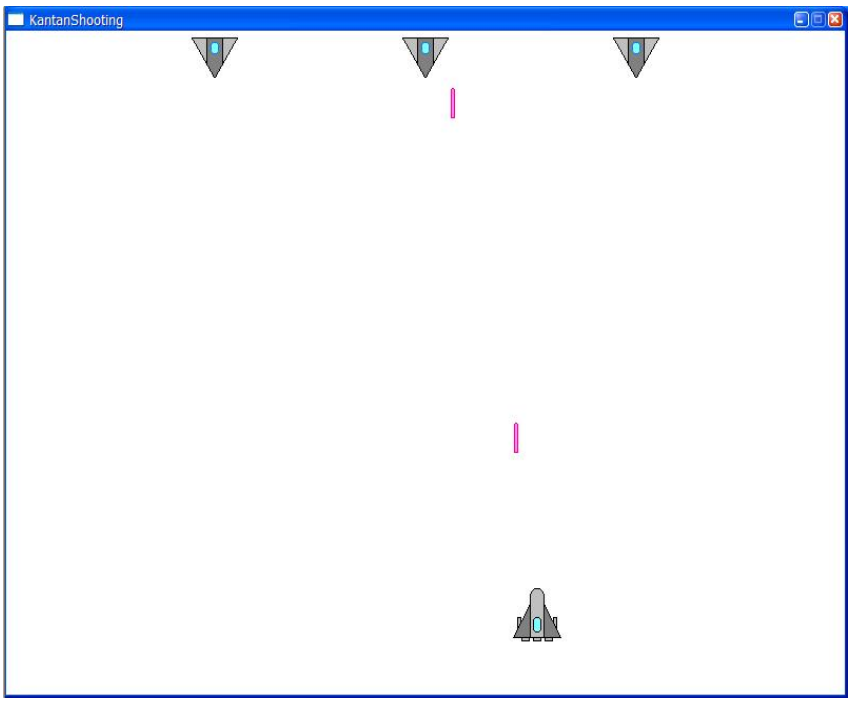
第 1 回 コンパイルをしてみよう

始めに

ソフトゼミ A との違い

ソフトゼミ A では、基本構文を学ぶために、コマンド プロンプト上で動くプログラムを作りますが、このソフトゼミ B では普段よく見かけるタイプの、ウインドウ上に画像などを表示するゲームを作成していきます。

このゼミでは、理解することよりも、実際に手を動かして、プログラミングがどうものか体験することに意義があります。



【図 1-1】サンプルプログラム

第 1 回から第 5 回まで、少しずつソースコードに手を加えて、【図 1-1】のような簡単なゲームを作っていきます。余裕がある人は、自分で手を加えてオリジナリティ溢れる作品に仕上げて下さいい。

また、各自のレベルに合わせてゼミを進めるので、自信が無い人も安心して下さい。分からないところがあれば、個別指導で丁寧に教えます。

コンパイルをしてみる

無料で使えるコンパイラがいくつかあります。以下のコンパイラと開発支援ソフトは全て部室の PC に導入してあります。コマンドを毎回入力するのが面倒な場合は Visual Studio や BCC Developer、ここでは紹介していない make や Eclipse を使うことをお薦めします。

以下の 4 通りのうち、好きな方法を選んでコンパイルしてみて下さい。コンパイルができたら、「実行してみる」に進んで下さい。

BCC でコンパイル

Borland C++ Compiler（BCC）は Borland（2006/03/01 現在）が無償で公開している C++ コンパイラです。Borland のウェブサイトから、無料でダウンロードできます。（書籍の付録にも入っています）インストール方法は、「参考になるサイト」の「猫でもわかるプログラミング」を参照して下さい。

Borland C++Compiler 5.5 無償ダウンロード
http://www.borland.com/jp/products/cbuilder/freecompiler.html

コマンドプロンプトで、cd コマンドを使い、配布したソースコードがあるディレクトリに移動し次の黒字の部分を入力します。

正しくコンパイルができると、KantanShooting.exe が作成されます。

```
D:¥Projects¥KantanShooting¥BCC>bcc32 -W -c game.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
game.c:

D:¥Projects¥KantanShooting¥BCC>bcc32 -W -c main.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
main.c:
警告 W8057 main.c 112: パラメータ 'lpszCmdLine' は一度も使用されない(関数 WinMain )

D:¥Projects¥KantanShooting¥BCC>bcc32 -W -eKantanShooting.exe game.obj main.obj
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
```

「窓の手」というソフトを使うと、フォルダのアイコンを右クリック→「コマンドプロンプト」で、そのフォルダをカレントディレクトリとした状態でコマンドプロンプトを起動できるので便利です。

「窓の手」公式サイト
http://www.asahi-net.or.jp/~vr4m-ikw/

BCC Developer でコンパイル

BCC Developer は、コマンドプロンプトを使わずに BCC でコンパイルできるフリーソフトです。詳しい使い方は、「参考になるサイト」の「猫でもわかるプログラミング」を参照して下さい。

以下の URL からダウンロードできます。

Jm HomePage
http://www.hi-ho.ne.jp/jun_miura/

「新規作成」からプロジェクトを作成します。「ディレクトリ名」に保存する場所、「プロジェクト名」に「KantanShooting」と入力し「OK」をクリックします。Visual Studio 同様、「ディレクトリ名」で指定場所にできた KantanShooting フォルダに配布した game.c と main.c、game.h をコピーしておきましょう。

次に、「プロジェクト」→「プロジェクトに追加」をクリックして、game.c と main.c、game.h を開きます。「プロジェクト」→「プロジェクト設定」をクリックし、「Windows アプリケーション (-W)」にチェックを入れ「設定」をクリックします。

これで準備完了です。次回以降はこれまでの操作は必要ありません。「プロジェクト」→「メイク」をクリックすると、コンパイルが始まります。

GCC でコンパイル

the GNU Compiler Collection (GCC) は、C/C++を始めとする、様々な言語に対応したコンパイラ群です。オープンソースで、当然無料です。Linux では標準的なコンパイラです。(一部の学科の授業で使われています) GCC を Windows 上で使えるようにした MinGW があります。一応書いておきますが、今回のサンプルプログラムは Windows 上でしかコンパイルできません。

コマンドプロンプトで次の黒字の部分を入力します。

```
D:¥Projects¥KantanShooting>gcc -c main.c
D:¥Projects¥KantanShooting>gcc -c game.c
D:¥Projects¥KantanShooting>gcc -o KantanShooting.exe main.o game.o -lwinmm -lgdi32
```

Visual Studio 2005 でコンパイル

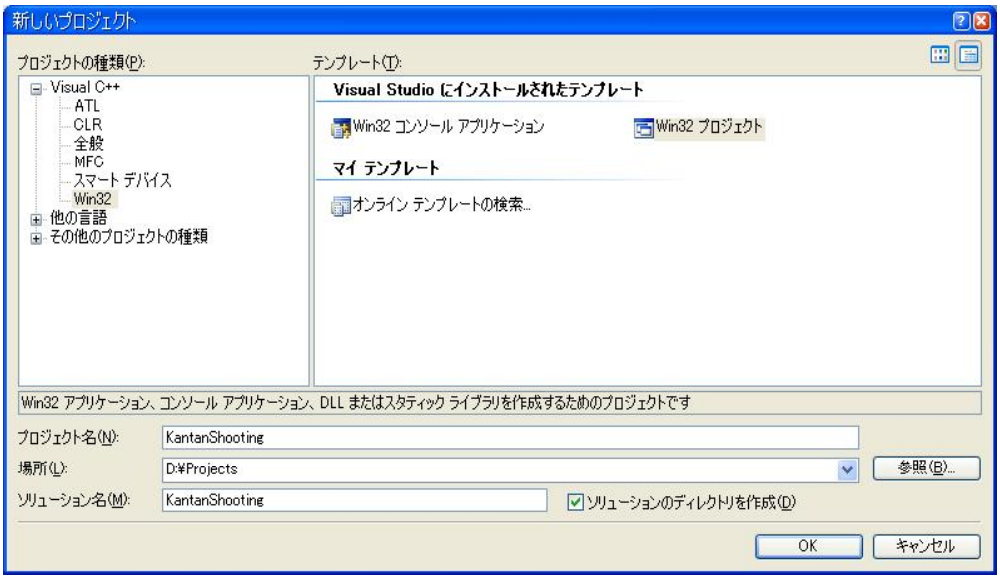
Visual Studio 2005 の Express 版は Microsoft のウェブサイトからダウンロードしたり、雑誌の付録からインストールすることができます。Standard 版のアカデミック版学割は 5000 円程度で購入できます。説明には Professional 版を使用しているので、前者とは若干異なる場合があります。

※補足

Express 版を使用する場合、別途 PlatformSDK をダウンロードする必要があります。

Visual Studio 2005 を起動し、「新規作成」→「新規プロジェクト」をクリックします。プロジェクトの種類で「Visual C++」→「Win32」を選び、テンプレートで「Win32 プロジェクト」を選びます。【図 1-2】

ウインドウ下部の「プロジェクト名」に「KantanShooting」と入力し、「場所」にプロジェクトファイルを保存したい場所を入力します。「場所」で指定した場所に KantanShooting フォルダが作成されているので、そこに配布した配布した game.c と main.c、game.h をコピーしておきましょう。



【図 1-2】新規プロジェクト

「OK」をクリックした後に出てくるウインドウで、「次へ」をクリックした後、【図 1-3】のように、「空のプロジェクト」にチェックを入れて「完了」をクリックします。

【図 1-3】空のプロジェクトにチェック

ソリューションエクスプローラで（表示されていない場合は、「表示」→「ソリューションエクスプローラ」をクリック）「ソースファイル」を右クリックして、「追加」→「既存の項目」をクリックします。【図 1-4】配布した、game.c と main.c を開いて下さい。

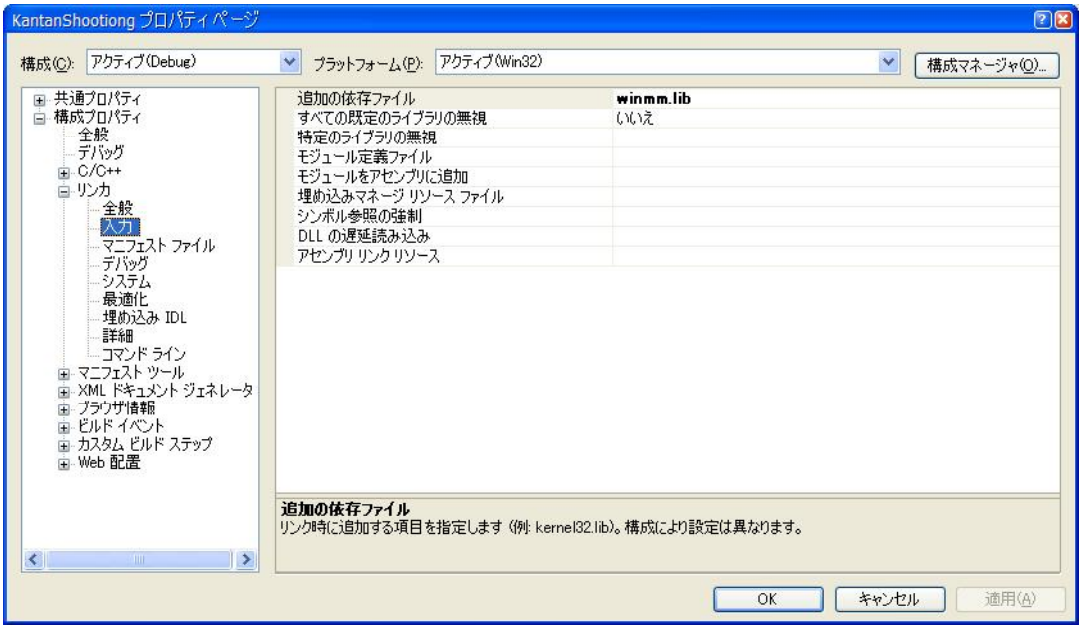
【図 1-4】ファイルを追加する

「ヘッダーファイル」を右クリックして、同様に game.h を開きます。そうすると、【図 1-5】のようになります。

【図 1-5】準備完了

次に、プロジェクトの設定をします。「プロジェクト」→「プロパティ」をクリックします。「構成プロパティ」→「リンカ」→「入力」の追加の依存ファイルに winmm.lib と入力し、「OK」をクリックします。【図 1-6】保存すれば準備完了です。

次回以降は保存したプロジェクトをダブルクリックで開くだけで、これまでの作業は一切必要ありません。



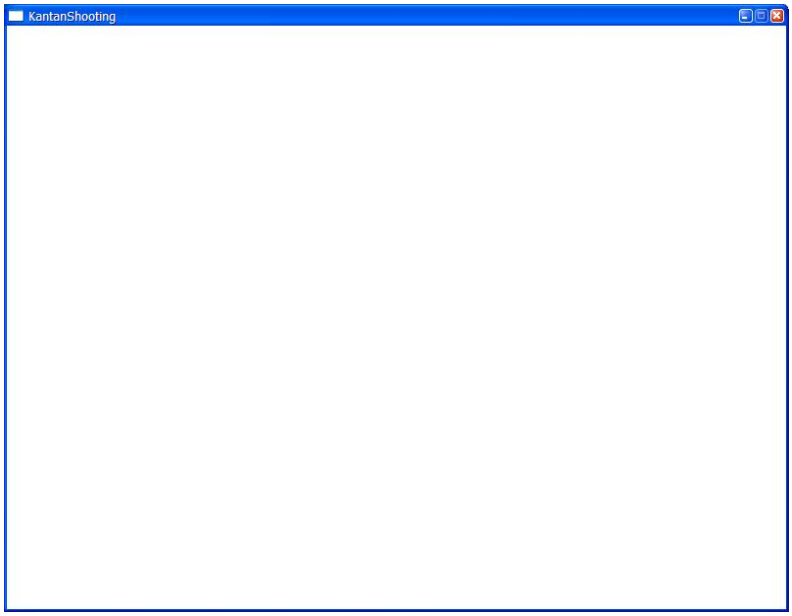
【図 1-6】プロジェクトのプロパティ

「ビルド」→「ソリューションのビルド」をクリックすると、コンパイルが始まります。実行する前に、配布した images.bmp を、作成した実行ファイル（KantanShooting.exe）と同じフォルダ（debug フォルダ）に入れておいて下さい。（そうしないと何も表示されません）

ソースコードを編集するには、ソリューションエクスプローラで編集するファイルをダブルクリックします。

実行してみる

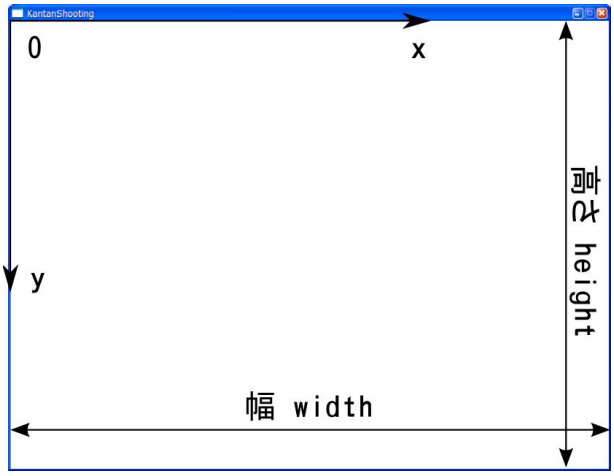
コンパイルが無事に成功すると、KantanShooting.exe が出来上がります。（ファイルが出力される場所はコンパイラによって異なります）KantanShooting.exe をダブルクリックすると、真っ白なウインドウが表示されます。これがこれから制作するゲームの原型です。



【図】真っ白なウインドウ

画面に自機を表示する

画面上の座標



【図 1-7】ウインドウ上の座標

ウインドウ上の座標は【図 1-7】のように、左上を原点として右下方向に進んで座標が大きくなっていきます。今回のサンプルプログラムでは、幅 800 ピクセル（ドット）、高さ 600 ピクセルの大きさのウインドウを使用します。

ビットマップファイル

配布したファイルの中に、images.bmp というファイルがあります。ペイント等で開いて中身を確認して下さい。（絵の善し悪しは突っ込まないこと。我慢できない場合は自分で書き換えて下さいな。）

画像が少ない時は、ビットマップ形式の画像を使うのが便利です。サウンドノベルやシミュレーションになると、JPEG や PNG といった方法で圧縮する必要がありますが…。

配布した画像ファイルは自機、敵機、弾の画像を 1 つの画像ファイルにまとめていますが、これはよく使われるテクニックです。

BitBlt 関数

さて、画像ファイルから、自機の部分を画面に表示してみましょう。ビットマップを読み込む処理は既にソースコードに書いてあるので、今回は読み込んだビットマップの部分領域を画面に転送する方法を紹介します。

読み込んだビットマップを画面に転送する方法として、最もシンプルな方法は BitBlt 関数を使う方法です。例えば、ビットマップファイルの座標(100,50)から、32x48 ピクセルの大きさの領域を、画面上の(300,200)に転送するには次のようにします。DC や SRCCOPY という単語が出ていますが、今は気にしないで下さい。

```
BitBlt(hdc, 300, 200, 32, 48, hbmpdc, 100, 50, SRCCOPY);
```

つまり、BitBlt 関数の引数は次のようになっています。

```
BitBlt(描画先 DC, 描画先位置 x, y, 幅, 高さ, 表示する画像 DC, 範囲の左上 x, y, オプション);
```

本日の作業

では、早速ソースコードに書き込んでみましょう。game.c をエディタで開いて、次の黒字の部分を入力して下さい。(灰色の部分は既にソースコードに書かれている部分です)

「/*」と「*/」の間に書かれた部分はコメントと言い、プログラムをする上でのメモ書きですので、入力しなくても構いません。

```
/* Draw 関数 画面の描画を行う */
/* 引数 hdc 描画先のデバイスコンテキストハンドル */
void Draw(HDC hdc)
{
	/*ここに画面に表示する処理を書く*/

	/*機体を表示する*/
	/*描画先 DC, 描画先位置 x, y, 幅, 高さ, 表示する画像 DC, 範囲左上 x, y*/
	BitBlt(hdc, 0, 0, 48, 48, hbmpdc, 0, 0, SRCCOPY);

	return;
}
```

入力したら、保存してコンパイルしてみましょう。実行して、画面左上に自機が表示されたら完成です。

最後に

Q&A その 1

Q. タイピングが遅いのですが、どうすればいいのでしょうか？
A. タイピング練習ソフトを使って練習するのもアリですが、チャットをやってみることをお薦めします。同期や先輩からメールアドレスを聞いて、MSN Messenger を始めてみてはどうでしょうか。

Q. プログラミングって難しいですか？
A. 学祭の前にちょっとやれば、すぐに作品ができるというものではありません。頑張って、PC を自在に操って、自分のアイデアをカタチにできるようになって下さい。

Q. パソコンを買おうと思っているのですが、どれを買えばいいのか分かりません。
A. 先輩に相談してみましょう。きっと具体的なアドバイスを得られると思います。

サークルを利用して!!

このソフトゼミを受けている人のほとんどは、プログラミングは始めてではないかと思います。最初は誰もが Lv0、経験値 0 からのスタートです。現実には「はぐれメタル」のようなオイシイ話はありませんが、それなりに経験を積んだ先輩等の仲間はいます。独りで戦うよりは少し効率が良いのではないでしょうか？

先輩や仲間と交流する場として、部室があります。昼休み、授業後など、是非部室に来て、先輩や機材を有効活用して下さい。特に、部室の PC には、プログラミングに必要なソフトが一通り入っています。

部室で昼食を取るもよし、ゲーム、漫画、麻雀… etc をするも…。ノート PC でコンパイラを導入の仕方が分からない人は、是非、部室に PC を持ってきて下さい。

参考になるサイト

自学自習用に参考になるサイトを紹介しておきます。

猫でもわかるプログラミング
http://www.kumei.ne.jp/
BCC、BCC Developer の導入方法、C 言語基礎、WindowsAPI の説明など。

第 2 回 変数で座標を管理

変数で表示位置を変える

変数で表示位置を指定する

前回は、自機を(0,0)の位置に表示しました。その方法は BitBlt 関数に(0,0)という座標を指定していましたが、その座標の指定に変数を使うこともできます。自機の位置を保持する変数 x と y があるとすると、次のようにすることで座標(x,y)に自機を表示できます。

```
int x;
int y;

… 省略 …

BitBlt(hdc, x, y, 48, 48, hbmpdc, 0, 0, SRCCOPY);
```

座標を管理する変数を宣言する

まず、前回書き加えたソースコードを開いて下さい。ソースコードの先頭に次の黒字の部分を書き足します。x と y はこのソースコードのあらゆる場所から呼び出すことができるグローバル（大域）変数です。

```
/*Windows の関数（WindowsAPI）を使うときはインクルードする*/
#include <windows.h>
#include "game.h"

HDC hbmpdc;
HBITMAP hbmp;

/*第 2 回変数*/
int x, y;
```

変数を宣言したら、前回、Draw 関数内に書いた BitBlt 関数の第 2、3 引数を変数 x、y に書き換えて下さい。

変数は宣言しただけの状態では、不定な値が格納されています。このまま実行すると、画面に何も表示されない可能性があるので、InitGame 関数に x と y に 0 を代入する処理を加えて下さい。

自機をキーボード操作で動かす

動かす=座標を管理する変数の値を変える

では、自機を動かすにはどうすればいいでしょうか？ KantanShooting では Update 関数と Draw 関数が繰り返し呼び出されます。Draw 関数では、一度画面を消去し、BitBlt 関数で画像を表示します。BitBlt 関数では、変数 x,y を使って座標を指定しました。

つまり、自機を動かすには、キーボードが押された時に、先程宣言した変数 x と y の値を増やしたり、減らしたりすればいいのです。

キーボードが押された時

キーボードの判定や、座標の更新は Update 関数で行います。Update 関数の中に次のような箇所があります。よく分からない単語が沢山あるかと思いますが、気にしないで下さい。キーボードの「←」キーが押された時に、自機を左に動かすには、x 座標を管理する変数の値を減らします。（座標軸については第 1 回のテキストを参照）具体的には次のようにします。

黒字の部分を実際にソースコードに入力して、コンパイル、実行してみて下さい。

```
/*ここに更新（座標など）をする処理を書く*/
if (keystate[VK_LEFT] & 0x80)
{
        /*←キーが押された*/
        x = x - 1;
}
return;
```

（保存を忘れずに!!）自機が左に動きますか？

本日の作業

「↑」キーを押したら上へ、「↓」キーを押したら下へ動くように、ソースコードを編集してみて下さい。出来上がったら、コンパイルをして正しく動作するか試してみましょう。

```
if (keystate[VK_LEFT] & 0x80)
{
	/*←キーが押された*/
	x = x - 1;
}
if (keystate[VK_RIGHT] & 0x80)
{
	/*→キーが押された*/
	変数の値を変える
}
if (keystate[VK_UP] & 0x80)
{
	/*↑キーが押された*/
	変数の値を変える
}
if (keystate[VK_DOWN] & 0x80)
{
	/*↓キーが押された*/
	変数の値を変える
}
return;
```

おまけ

余裕のある人は、応用して、スペースキーが押された時に、画面中央に瞬間移動するように改造して下さい。

ヒント:スペースは VK_SPACE

補足

このキーボードが押された時の処理（Update 関数）は、1 秒間に 60 回（60FPS）呼び出されるようになっています。今回のプログラムではキーを押し続けると 1 秒間に 60 ピクセルの速さで自機が動きます。

最後に

Q&A その 2

Q. ソースコードを編集する時にどのソフトを使えばいいですか？

A. メモ帳でも構いませんが、専用のエディタの方が色分けなどの機能があり便利です。Visual Studio や BCC Developer などの開発環境を使っている場合は、付属のエディタを使えばいいでしょう。Visual Studio は入力補完があって便利です。そうでない場合は、（このテキストの作者的には）TeraPad や HeTeMuLuCreator がお薦めです。

TeraPad（フリーソフト）
http://www5f.biglobe.ne.jp/~t-susumu/
HeTeMuLu Creator（フリーソフト）
http://www.hosiken.com/
EmEditor（一部フリーソフト）
http://www.emurasoft.com/jp/
サクラエディタ（フリーソフト）
http://sakura-editor.sourceforge.net/
秀丸エディタ（学生は登録すれば無料で使える）
http://hide.maruo.co.jp/

～(市販 or 同人ゲーム名)みたいなゲームは作れますか？

Q&A の番外編です。プログラミングには材料は必要ありません。道具（コンパイラとか）さえあれば、(時間が無限大にあれば) 何でも作れるはずです。

少々話が逸れますが、例えばテニスサークルの人たちは、週に何回か 1 日数時間練習をします。最初はラケットにボールを当てることさえできなかった人でも、2 年、3 年と練習を積み重ねれば試合に勝てるようにまで上手くなります。

プログラムも同じです。今、何かに使っている時間の少しをプログラミングに回してみて下さい。少しずつ経験を積めば、来年、再来年には同人ゲームとして販売できる程の作品を作れるようになっているかも？

実際に手を動かすことが大切!!

プログラミングは、構文を勉強したり、本を読んだりしただけでは身に付きません。2 回のソフトゼミを通して、色々と構文等を覚えたと思いますが、必ず自分の手を動かして、プログラムを打ち込み、コンパイルし実行してみて下さい。

プログラムをちょっと改造したりした時は、それがどんなに些細な物でも、是非部室に持ってきてみんなに見せましょう。話のネタにもなるし、作って満足して終わりにするよりも、モチベーションが長続きするのではないでしょうか？

第３回 if文で画面から出ないように

自機が画面の外に出てしまう

if文で範囲を制限する

前回、作成した実行ファイルを実行してみて下さい。自機をある方向に動かし続けると、自機がウインドウの外に出てしまうことが分かると思います。これを防ぐには、if 文を使って画面から出た時に、ウインドウ内に戻すようにします。

ウインドウ上端から出ないようにするには次のようにします。(x,y は機体の左上の座標であることに注意して下さい。)

```
void Update(byte* keystate)
{

…省略…

…移動する処理…

        if (y < 0)
        {
                y = 0;
        }

        return;
}
```

前回編集したファイルを開いて、黒字の部分を入力し、上端から出ないことを確認して下さい。

本日の作業

上端の場合と同様に、画面から自機が出ないようにソースコードを編集し、コンパイル、実行してみて下さい。

おまけ

1. 画面の端に移動すると、画面の反対側に移動するように改造して下さい。

2. 自機の速さを 2 にして、Ctrl キーを押している間は速さが 1 になるように改造して下さい。ヒント、Ctrl キーは VK_CONTROL です。

第 4 回 配列と for 文で敵を表示

敵機体の準備

enemy1 、enemy2 …なんて変数はメンドクサイ!!

```
/*第 4 回配列と For 文*/
int enemy_x_1;
int enemy_y_1;

int enemy_x_2;
int enemy_y_2;

int enemy_x_3;
int enemy_y_3;
```

敵の座標をこのように管理していると、様々な問題があります。敵の座標を更新するのに、各敵毎に処理を書かなければいけなかったり、敵の数の変化にも対応できません。このように、同じものを複数管理するには配列を使うのが一般的です。

配列を使う

```
/* Windows の関数（WindowsAPI）を使うときはインクルードする*/
#include <windows.h>
#include "game.h"

HDC hbmpdc;
HBITMAP hbmp;

/*第 2 回変数*/
int x, y;

/*第 4 回配列と For 文*/
int enemy_x[3]; /*敵の x 座標*/
int enemy_y[3]; /*敵の y 座標*/
```

さて、配列を使って、敵の座標を管理する変数を宣言してもました。上記の例よりもすっきりしていませんか？早速、早速黒字の部分をソースコードに入力して下さい。

for 文でまとめて

for 文と配列

配列と必ずといっても組み合わされるのが for 文です。for 文と配列で次のようなコードがしばしば使われます。

```
int a[100];

for (i = 0; i < 100; i++)
{
        … 計算とか …
        a[i] = … 式 …
}
```

上記の例の配列 a の最後の添字は 99 です。間違えないようにしましょう。

初期位置の設定

まずは、敵の初期位置を設定してみましょう。ゲームの初期化を行う InitGame 関数を次のようにします。ソースコードに次の黒字の部分を書き込んで下さい。

自機の初期位置は中央に設定しておきましょう。

```
/* InitGame ゲームの初期化を行う*/
void InitGame(void)
{
        int i;
        /*ここにゲームを初期化する処理を書く*/

… 省略 …

        /*機体の初期位置*/
        x = 276; y = 400;

        /*敵機体の初期位置*/
        for(i = 0; i < 3; i++)
        {
                enemy_x[i] = 200 - 24 + 200 * i;
                enemy_y[i] = 0;
        }
        return;
}
```

本日の作業

敵の表示も自機の表示と同じように BitBlt 関数を使います。ビットマップファイル上の敵画像の座標は配布した images.bmp を開いて確認して下さい。敵の画像の大きさは 48 × 48 ピクセルです。

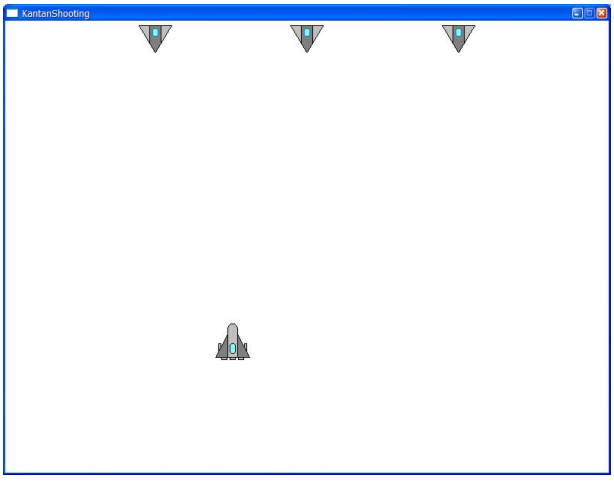
```
/* Draw 関数画面の描画を行う*/
/* 引数 hdc 描画先のデバイスコンテキストハンドル*/
void Draw(HDC hdc)
{
	ループ変数の宣言

	/*ここに画面に表示する処理を書く*/

	/*機体を表示する*/
	/*描画先 DC, 描画先位置 x, y, 幅, 高さ, 表示する画像 DC, 範囲左上 x, y*/
	BitBlt(hdc, x, y, 48, 48, hbmpdc, 0, 0, SRCCOPY);

	/*敵機体を表示する*/
	for (ループ条件)
	{
		i 番目の敵の表示
	}
	return;
}
```

実行して敵が表示されるか確認して下さい。



【図 4-1】敵の表示

おまけ

敵の初期位置を自分の好きなように変えてみて下さい。

第 5 回 構造体でひとまとめ＆弾を発射!!

構造体でひとまとめ

構造体

敵が持つ情報にはどのようなものがあるでしょうか？ X 座標、Y 座標、HP など様々な情報を持っています。前回のように、配列で各情報をバラバラに保持することもできますが、構造体を使うと、よりすっきりさせることができます。

敵の情報

敵の情報は次のような構造体の配列で管理します。

```
/*第 5 回構造体*/
struct ENEMY
{
        int x;
        int y;
        int HP;
};
```

弾の情報

弾を発射できるようにします。弾の情報を次のように宣言します。コメントにも書いてありますが、x と y が INT_MIN の時に、その弾は使われていないとすることにします。

INT_MIN はコンパイル時に、int 型の最小値に自動的に置き換えられます。

```
/*(INT_MIN, INT_MIN)の時は空を表す*/
struct BULLET
{
        int x;
        int y;
};
```

変数を宣言する

構造体は宣言しただけでは使えません。変数を宣言します。

```
struct ENEMY enemys[3];
struct BULLET bullets[20];
```

弾を発射

「Z」キーが押された時に弾を発射するようにします。キーが押された時の処理の書き方は第 2 回で説明した通りです。

```
/*INT_MIN を使うために必要*/
#include<limits.h>
/*次に使う buletts 配列の添字*/
int bullets_i = 0;
/*弾と弾の発射間隔制御用*/
int bullet_wait = 0;

…省略…

void Update(byte* keystate)
{

	…省略…

	if (keystate['Z'] & 0x80)
	{
		/*Z キーが押された*/
		if (bullet_wait >= 30)
		{
			bullets[bullets_i].x = x;
			bullets[bullets_i].y = y - 48;
			bullets_i = ++bullets_i % 20;
			bullet_wait = 0;
		}
	}

	…省略…

	/*弾を動かす*/
	/*弾の位置は y 座標でソートされているので、もっと速いアルゴリズムがあるけど、遅い方で*/
	for (i = 0; i < 20; i++)
	{
		/*画面外に出た弾の処理*/
		if (bullets[i].y < -48)
		{
			bullets[i].y = INT_MIN;
			bullets[i].x = INT_MIN;
		}
		else
		{
			bullets[i].y -= 10;
		}
	}
	/*前の弾を発射してからの時間を更新*/
	bullet_wait++;

	return;
}
```

本日の作業

ソースコードに各部分の黒字の部分を入力し、前回作成した enemy_x、enemy_y 配列の宣言を削除して、enemys 配列に置き換えて下さい。

InitGame 関数内に弾を初期化する処理を、Draw 関数内に弾を表示する処理を書き加え、コンパイルして実行してみて下さい。この時、弾を動かす処理を参考にして使用していない弾を BitBlt で転送しないようにしましょう。

正常に動作したら、プログラムの流れを考えてみましょう。

おまけ

1. 今回のプログラムでは弾を画面上に 20 個までしか存在させることができません。30 個の弾を存在可能にするにはどうすればいいでしょうか？(考えるだけで良い)

2. 弾の発射間隔が長くなるように改造して下さい。

第 6 回 当たり判定

プリプロセッサ

プリプロセッサを使うと、コンパイルを行う前に何かをする場合に使います。そのうちの 1 つ#define（マクロ定義）は、コンパイル前に文字列の置き換えをします。次の例では

「LINE」を「printf(″define で置き換え¥n″);」に

「ARRAY_SIZE」を「10」に

置き換えます。

```
#include <stdio.h>

#define LINE printf("define で置き換え¥n");
#define ARRAY_SIZE 10

int main(void)
{
        int a[ARRAY_SIZE];
        int i;

        LINE
        for (i = 0; i < ARRAY_SIZE; i++)
        {
                scanf("%d", &a[i]);
        }
        for (i = 0; i < ARRAY_SIZE; i++)
        {
                printf("%d¥n", a[i]);
        }
        return(0);
}
```

実行すると次のようになります。

```
define で置き換え
1 3 5 6 7 9 8 7 6 5        ← 入力
1 3 5 6 7 9 8 7 6 5
```

#define の他にも#include や#if、#endif、マクロ関数などがあります。

仕様を確認しておこう

さて、次の作業に移る前に、今回の作業に関連したゲームの仕様を確認しておきましょう。

敵の数は常に 4。

敵の HP は 2。

敵は Y 軸方向に速さ 2 で動く。

敵が倒されたら、画面上端から新たな敵が出現する。X 座標はランダムな値とする。

同様に、敵が画面下端から消えた場合も画面上端から新たな敵が出現する。

敵を動かす

自機の移動や弾の移動と同様です。Update 関数に次のような処理を加えて下さい。

```
void Update(byte* keystate)
{

        …省略…

        /*敵を動かす*/
        for (ループ条件)
        {

                /*敵が画面から出た場合の処理*/
                if (敵が画面から出たら)
                {
                        新たな敵の出現（詳細は後述）
                }
        }
        return;
}
```

当たり判定

矩形

矩形（長方形）を表現するときには、しばしば RECT 構造体を使います。RECT 構造体は windows.h の内部（正確には windef.h）で次のように宣言されています。

```
typedef struct tagRECT
{
    LONG    left;
    LONG    top;
    LONG    right;
    LONG    bottom;
} RECT;
```

ソフトゼミ A で習った構造体の宣言と違って、typedef 宣言がありますが、これは struct tagRECT 型に RECT 型という別名を付けるということを意味します。矩形を表現する構造体の変数を宣言するには次のようにします。

```
RECT rc;
/*rc の left（左端の X 座標）に 300 を代入*/
rc.left = 300;
/*rc の bottom（下端の Y 座標）に 100 を代入*/
rc.bottom = 100
```

構造体は次のように初期化することができます。

```
RECT rc = {10, 36, 20, 52};

/*
rc.left     に 10
rc.top      に 36
rc.right    に 20
rc.bottom   に 52
が代入される
*/
```

補足 一般的には right の列と bottom の行は、領域に含まれないと考えるのが一般的ですが、今回は領域に含まれると考えます。

アルゴリズム

いよいよ、シューティングで一番重要な当たり判定です。KantanShooting では矩形による当たり判定を行います。つまり、敵機の矩形と弾の矩形が重なっていれば、敵機に弾があたっていることになります。自機、敵機、弾の矩形は次のように設定しました。(塗りつぶした領域)

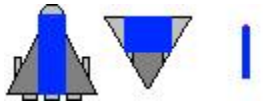


図 6-1

プログラム上ではこの領域を次のような変数で管理します。

```
/*画像上の当たり領域*/
RECT bullet_rc = {22, 10, 25, 35};
RECT my_rc = {17, 5, 14, 40};
RECT enemy_rc = {12, 6, 35, 23};
```

さて、矩形と矩形が重なっているかを判断するにはどのようにすれば良いでしょうか？次の図を見て下さい。

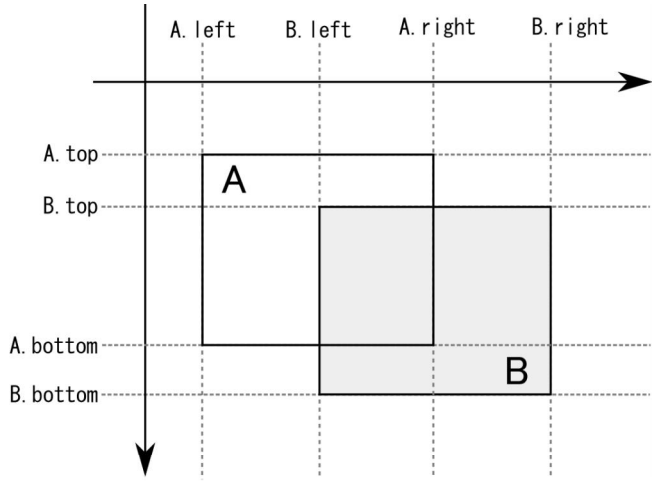


図 6-2 当たり判定アルゴリズム

矩形 A と矩形 B が重なりは次の式で判定できます。

```
X 軸方向に関して　A.left < B.right かつ B.left < A.right
Y 軸方向に関して　A.top < B.bottom かつ B.top < A.bottom
```

C 言語で表現すると次のようになります。

```
if(A.left < B.right && B.left < A.right && A.top < B.bottom && B.top < A.bottom)
{
        矩形 A と矩形 B が重なっている
}
```

当たり判定を行う関数

次のような当たり判定を行う CheckHit 関数を作成して下さい。CheckHit 関数は Update 関数の最後で呼び出します。

```
/* CheckHit 関数当たり判定を行う*/
/* 引数なし*/
void CheckHit()
{
	int i, j;
	RECT brc; /*弾の実座標*/
	RECT erc[敵の数]; /*敵の実座標*/

	/*敵の実座標の設定*/
	for (j = 0; j < 敵の数; j++)
	{
		erc[j] = enemy_rc;
		erc[j].left += enemys[j].x;
		erc[j].right += enemys[j].x;
		erc[j].top += enemys[j].y;
		erc[j].bottom += enemys[j].y;
	}

	/*総当たり当たり判定プログラム自機の弾→敵*/
	for (i = 0; i < 弾の数; i++)
	{
		/*実座標の設定*/
		brc = bullet_rc;
		brc.left += bullets[i].x;
		brc.right += bullets[i].x;
		brc.top += bullets[i].y;
		brc.bottom += bullets[i].y;

		if (bullets[i].x != INT_MIN && bullets[i].y != INT_MIN)
		{
			for (j = 0; j < 敵の数; j++)
			{
				/*当たり判定*/
				if(当たり判定条件)
				{
					enemys[j].HP--;
					/*敵を倒した場合の処理*/
					if (enemys[j].HP <= 0)
					{
						/*新しい敵の作成*/
						新しい敵の出現（詳細は後述）
					}
					敵に当たった弾を消す

					break; /*break 文…直前のループから抜ける*/
				}
			}
		}
	}
	return;
}
```

新たな敵の出現

乱数

ランダムな位置に敵を出現させたい…という場合には乱数を使います。今回は手軽な C 言語の標準関数である rand 関数を使います。rand 関数は 0 ～ RAND_MAX までの値を返します。例えば、サイコロの目を出力するプログラムは次のようになります。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
	int i;
	srand(time(NULL));
	for (i = 0; i < 10; i++)
	{
		printf("サイコロの目: %d¥n", rand() % 6 + 1);
	}
	return 0;
}
```

このようにして乱数を使う場合は、stdlib.h と time.h をインクルードして下さい。また

srand(time(NULL));

は乱数の初期化です。プログラムの初期化時に 1 回だけ記述して下さい。(書かないと rand 関数が毎回同じ順番で値を返します)

敵を作成する関数

敵の HP が 0 になった時と、敵が画面から出た時に新たな敵を作成するので、同じ事を 2 度書かずに済むように CreateEnemy 関数を作ります。

```
/* CreateEnemy 関数新たな敵を生成する*/
/* 引数 ei 作成する敵番号*/
void CreateEnemy(int ei)
{
	enemys[敵番号].HP = 最大 HP;
	enemys[敵番号].x = 画面全体にランダムに出現;
	enemys[敵番号].y = - 画像の大きさ;
	return;
}
```

本日の作業

1. ソースコード中の定数(数値)を次のマクロ定義で置き換えて下さい。また当たり判定用の矩形を保持する変数を宣言して下さい。

```
/*敵の数*/
#define NUM_ENEMYS 4
/*bullets 配列の大きさ*/
#define MAX_NUM_BULLETS  20
/*転送する領域のサイズ*/
#define IMAGE_SIZE 48
/*画面の幅*/
#define SCREEN_WIDTH 800
/*画面の高さ*/
#define SCREEN_HEIGHT 600
/*敵の最大 HP*/
#define ENEMY_MAX_HP 2
/*弾の発射間隔*/
#define BULLET_WAIT_LENGTH 30
/*弾の速度*/
#define BULLET_V -10
/*敵の動く速さ*/
#define ENEMY_V 1

/*画像上の当たり領域*/
RECT bullet_rc = {22, 10, 25, 35};
RECT my_rc = {17, 5, 14, 40};
RECT enemy_rc = {12, 6, 35, 23};
```

2. 敵の移動処理（Update 関数）、CheckHit 関数、CreateEnemy 関数を完成させて下さい。（srand()を忘れないように注意）
3. InitGame 関数に敵の HP を初期化する処理を追加して下さい。
4. 敵の初期位置などを調整してコンパイル、実行してみましょう。

おまけ

ランダムで動きが速い敵が出現するように改造しみよう。

ヒント:ENEMY 構造体に速度を保持するメンバー変数を追加

第 7 回 文字列の表示

仕様を確認しておこう 2

仕様

敵は平均 2 秒に 1 回弾を発射する。

弾は自機方向に向けて発射する。

敵の弾の速さは 2。

自機の最大 HP は 20。

敵 1 機を撃退したときのスコアは 100。

マクロとグローバル変数、構造体

```
/*第 7 回 sqrt に使う*/
#include <math.h>

int HP;

/*enemybullets 配列の大きさ*/
#define MAX_NUM_ENEMYBULLETS 30
/*敵の弾の速さ*/
#define ENEMYBULLET_V 2.0
/*敵を機撃退したときのスコア*/
#define SCORE_PER_ENEMY 100
/*文字列出力用バッファサイズ*/
#define BUFFER_SIZE 100
/*最大 HP*/
#define MAX_HP 20;

/*敵の弾*/
struct ENEMY_BULLET
{
        double x;
        double y;
        double vx;
        double vy;
        int fUsing; /*使用しているか*/
};

struct ENEMY_BULLET enemybullets[MAX_NUM_ENEMYBULLETS];

/*次に使う enemybullets 配列の添字*/
int enemybullets_i = 0;

unsigned int framecount;
unsigned int score;
```

敵の弾

速度ベクトルを求める

自機方向の長さ 2 のベクトルを求めるにはどうすればよいでしょうか？敵機の座標を(x0, y0)、自機の座標を(x1, y1)とすると、自機方向のベクトルは(x1-x0, y1-y0)で求まります。このベクトルの大きさは

$$\sqrt{(x1 - x0)^2 + (y1 - y0)^2} \quad (=l \text{ とする})$$

なので、求めるベクトルは次の通りです。

$$\left((x1 - x0) / l \times \text{弾の速さ}, (y1 - y0) / l \times \text{弾の速さ}\right)$$

弾の発射

速度ベクトルを求める以外は、自機の弾の発射と基本的には同じです。自機の場合は弾を使っていない状態を表すために INT_MIN を使いましたが、今回は使用しているかどうかを保持するメンバー変数（フラグ）fUsing を使います。

sqrt 関数で引数の平方根を求めることができます。

```
void Update(byte* keystate)
{
    double vx, vy, veclength;

    …省略…

    /*敵の弾を発射*/
    for (i = 0; i < NUM_ENEMYS; i++)
    {
        if (! (rand() % 120))
        {
            enemybullets[enemybullets_i].fUsing = 1;
            /*自機に向けて発射する*/
            enemybullets[enemybullets_i].x = enemys[i].x;
            enemybullets[enemybullets_i].y = enemys[i].y;
            /*弾の向きの設定*/
            vx = x - enemybullets[enemybullets_i].x;
            vy = y - enemybullets[enemybullets_i].y;
            /*ベクトルの長さを計算*/
            veclength = 長さ;
            enemybullets[enemybullets_i].vx = ベクトルの X 成分;
            enemybullets[enemybullets_i].vy = ベクトルの Y 成分;
            enemybullets_i = ++enemybullets_i % MAX_NUM_ENEMYBULLETS;
        }
    }

    …省略…

    return;
}
```

敵の弾を移動

```
void Update(byte* keystate)
{

    …省略…

    /*敵の弾を動かす*/
    for (i = 0; i < MAX_NUM_ENEMYBULLETS; i++)
    {
        if (この弾を使っている)
        {
            /*画面内に弾があるか*/
            /*第 6 回矩形同士の当たり判定を参照*/
            if (enemybullets[i].x <= SCREEN_WIDTH && 0 <= enemybullets[i].x + IMAGE_SIZE &&
                enemybullets[i].y <= SCREEN_HEIGHT && 0 <= enemybullets[i].y + IMAGE_SIZE)
            {
                enemybullets[i].x += 座標を増やす;
                enemybullets[i].y += 座標を増やす;
            }
            else
            {
                enemybullets[i].fUsing = 0;
            }
        }
    }

    …省略…

    return;
}
```

画面にテキストを描画

文字列とは

C 言語で文章を扱うには文字列を使います。文字列とは、その名の通り文字の列で、実態は文字型(char 型)の配列です。printf 関数で使用したように、次のように「"」で囲んで書くことができます。

```
"文字列"
```

printf 関数に似た関数で sprintf 関数があります。sprintf 関数は第 1 引数に指定した文字型配列に対して、printf 関数の出力を書き込みます。

```
char buf[100];
sprintf(buf, "%d + %d = %d", 3, 4, 7);
```

ただし、Windows には国際化に対応するため UNICODE という文字コードをしばしば使います。文字コードというのは、文字に割り当てる ID 表のようなものです。この UNICODE では全ての文字を 2 バイトで表現します。Shift-JIS という文字コード（日本語版 Windows 環境で普通にプログラムを書くと Shift-JIS になります）もありますが、アルファベットは 1 バイト、日本語は 2 バイトで表現するので、文字単位で処理するときに非常に不便です。

UNICODE の文字は char 型の代わりに WCHAR 型を使います。UNICODE 文字列を操作するには、wprintf、wsprintf 等の専用の関数を使います。また UNICODE を使う場合、文字列は次のように表記します。

```
wprintf(TEXT("テスト%d¥n", 50));
```

また UNICODE を使うには#include<windows.h>の前で次のマクロを宣言して下さい。VisualStudio を使っている場合は、「KantanShooting のプロパティ」【図 1-6】で「全般」→「構成のプロパティ」→「文字セット」を「Unicode　文字セットを使用する」に設定すれば（おそらく、設定済みです)、マクロを定義する必要はありません。

```
#define UNICODE
```

その他、文字列に関しては C 言語の入門書で調べて下さい。

TextOut 関数

さて、ここからが本題です。printf 関数はコマンドラインに文字列を出力する関数でしたが、ウインドウ上に文字列を出力する TextOut 関数があります。TextOut 関数は次のように宣言されています。

```
BitBlt(描画先 DC, 描画先位置 x, y, 出力する文字列, 出力する文字数);
```

今回は現在の HP とスコアと経過時間を表示します。

```
void Draw(HDC hdc)
{
    WCHAR buf[BUFFER_SIZE];

    …省略…

    /*スコアと HP の表示*/
    wsprintf(buf, TEXT("HP %3d  Score %6d  Time %4d.%1ds"),
             HP, score, framecount / 60, framecount / 6 % 10);
    TextOut(hdc, 500, 0, buf, lstrlen(buf));

    return;
}
```

lstrlen 関数は引数で指定された文字列の文字数を返します。UNICODE マクロを定義すると、UNICODE を扱うことがでる関数です。

本日の作業

1. 必要なマクロとグローバル変数、構造体を宣言して下さい。
2. 敵の弾の座標はなぜ double 型を使うのでしょうか？
3. 宣言した変数の初期化を行う処理を加えて下さい。
4. 敵の弾の発射する処理、動かす処理を完成させて下さい。
5. 敵の弾を表示する処理を完成させて下さい。(画像の大きさは 48 ピクセル)
6. ゲームを開始してからのフレーム数を framecount 変数でカウントする処理を加えて下さい。(Update 関数でインクリメントする)
7. この状態で正しく動作するか、コンパイルして確認して下さい。
8. HP とスコア、経過時間を表示する処理を加えて下さい。(UNICODE の設定を忘れずに)
9. 敵を 1 体倒すと、スコアを加算する処理を加えて下さい。

おまけ

敵の弾と自機の当たり判定を完成させて下さい。(前回のプログラムとほとんど同じです)

第 8 回 最後の総まとめ

main.c を開けてみると…

WindowsAPI とは

Windows の機能を呼び出す関数を Windows API と言います。今までに何度か使った BitBlt 関数や TextOut 関数も Windows API です。Windows API は windows.h で定義されています。

WndMain 関数とウインドウの作成

ウインドウを使うプログラムを作るときに、BCC では bcc32 の後に-W というオプションを付けましたが、この時プログラムは main 関数からではなく、WinMain 関数から始まります。

ウインドウを作成する処理を簡単に説明します。ウインドウを作成する処理は少々長いのですが、どんなプログラムを作るときも共通なので、コピー＆ペーストすれば大丈夫です。main.c の WinMain 関数を見て下さい。ウインドウを作成する手順は次の通りです。

1. HWND 型の hWnd 変数と WNDCLASS（構造体）型の wc 変数、RECT 型の変数 cr を宣言する。
2. wc にパラメーターを設定する。
3. RegisterClass 関数でウインドウクラス(wc)をシステムに登録数する。
4. AdjustWindowRect 関数でウインドウの枠などを含めたウインドウの大きさを求める。
5. CreateWindow 関数でウインドウを作成する。
6. ShowWindow 関数、UpdateWindow 関数でウインドウを表示する。
7. SetFocus 関数でウインドウをアクティブにする。

この一連の処理で重要なのは、CreateWindow 関数の戻り値として hWnd 変数が受け取った値はウインドウハンドルと呼ばれる物で、作成したウインドウの ID です。このウインドウに対して処理を行う場合は、Windows API に hWnd の値を渡します。Windows ではしばしばハンドルと呼ばれる物でオブジェクト（ウィンドウ、画像、プリンター…）を識別します。

wc.lpfnWndProc で代入している WndProc は、プロトタイプ宣言されている WndProc 関数へのポインタと呼ばれる物で、詳しくは後述します。

メッセージループ

キーボードを押したり、マウスをクリックしたり、閉じるボタンを押したり、他のウインドウが重なたり…した時にWindowsからメッセージという物が送られてきます。

WinMain 関数から呼び出されている GameLoop 関数内には無限ループがあります。このループは game.c の Update 関数、DrawBackground 関数、Draw 関数を繰り返し呼び出しています。このループの最初で、PeekMessage 関数でメッセージが送られていないか調べ、TranslateMessage 関数、DispatchMessage 関数でメッセージを処理しています。その後、wc.lpfnWndProc で設定した関数（今回の場合は WndProc 関数）が呼び出されます。プログラムの終了が検知されたらループを抜け終了します。

タイマー

game.c 内の関数の前に if 文がありますが、この if 文は game.c 内の関数が 60FPS（16.666 …ミリ秒間隔）で呼び出されるようにするものです。timeGetTime 関数はWindowsが起動してからの経過時間をミリ秒単位で返す関数です。

起動時に timeBeginPeriod 関数を、終了時に timeEndPeriod 関数を呼ぶことでタイマーの精度をミリ秒単位で指定できます。

switch 文

switch 文は値によって動作を切り替えるときに使います。

```
int a;
scanf("%d", &a);

switch(a)
{
        case 1: /* a が 1 の時に実行される */
                printf("入力された値は 1 です。");
                break;
        case 2: /* a が 2 の時に実行される */
                printf("入力された値は 2 です。");
                break;
        default: /* どの条件にも該当しない場合に実行される */
                printf("入力された値は 1 でも 2 でもありません。");
}
```

break 文がないと、次の case の箇所まで実行してしまうので注意が必要です。例えば、case 1:の break 文がないと、a=1 の時に

```
入力された値は 1 です。
入力された値は 2 です。
```

と出力されてしまいます。

WndProc 関数

WinProc 関数は送られてきたメッセージを処理する関数です。メッセージの種類は引数 msg に渡されます。メッセージの種類には WM_DESTROY や WM_SETCURSOR、WM_ERASEBKGND の他に、WM_PAINT、WM_MOUSEMOVE などがあります。

WinProc 関数で重要なのは、WM_DESTROY を受け取ったときに PostQuitMessage 関数を呼び出すことと、最後に DefWindowProc 関数を呼び出すことです。DefWindowProc 関数は、ウインドウの最小化などを Windows に任せる時に使います。

ウインドウアプリケーションはムズカシイ!?

ここまでずらずらと難しげな文章が続いていましたが、分からなくても大丈夫です。必要なときに書籍等を見ながら書ければ十分です。このテキストを書くときも、MSDN ライブラリや Windows ゲームプログラミングを見ながら書いています。

実際に制作で使う場合には、一度、書籍やウエブサイトを見て勉強したほうが良いでしょう。

ビットマップとダブルバッファリング

ビットマップの読み込み

KantanShooting では images.bmp を読み込んで BitBlt 関数で転送していますが、この方法を紹介します。画像ファイルは game.c の InitGame 関数で読み込んでいます。

```
HDC hbmpdc;
HBITMAP hbmp;

/*画像を読み込む*/
hbmpdc = CreateCompatibleDC(NULL);
hbmp = LoadImage(0, TEXT("images.bmp"), IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
SelectObject(hbmpdc, hbmp);
```

LoadImage 関数の第 2 引数は読み込むファイル BitBlt 関数などでこの画像を操作する場合は、SelectObject 関数に渡した HDC 型の hbmdc を渡します。このビットマップファイルを使い終わったら必ず次のように解放して下さい。

```
/*読み込んだ画像を解放する*/
DeleteDC(hbmpdc);
DeleteObject(hbmp);
```

ダブルバッファリング

ビットマップは画像ファイルとして使うだけではなく、描画先の対象としても使うことができます。

ウインドウに対して、直接描画関数を呼び出すと、1 フレームに何度も描画が行われるので、画面がちらついてしまいます。そこで、一度メモリー上に作成したビットマップに描画し、最後にこのビットマップをウインドウに転送することにより、描画回数を減らし、ちらつきを防ぐことができます。

GetDC 関数は引数で指定したウインドウのデバイスコンテキストを返す関数です。

```
HDC hdc, hBackBufferDC;
HBITMAP hBackBuffer;

/*ダブルバッファリング（ちらつき防止対策）*/
hdc = GetDC(hWnd);
hBackBuffer = CreateCompatibleBitmap(hdc, 800, 600);
hBackBufferDC = CreateCompatibleDC(NULL);
SelectObject(hBackBufferDC, hBackBuffer);
```

hdc と hBackBufferDC、hBackBuffer は使い終わったら解放しましょう。

```
hdc = GetDC(hWnd);
DeleteDC(hBackBufferDC);
DeleteObject(hBackBuffer);
ReleaseDC(hWnd, hdc);
```

game.h

game.h を開いてみて下さい。game.h には game.c の関数のプロトタイプ宣言があります。game.h を main.c でインクルードすることにより、main.c から game.c の関数を呼び出せるようになります。

また、インクルードする時に、game.h が<>ではなく""で囲まれていますが、ソースファイルと同じディレクトリにあるヘッダーファイルをインクルードする場合に""を使います。

```
#include "game.h"
```

このように複数のソースコードに分けてコンパイルすることを分割コンパイルと言います。

本日の作業

Update 関数の戻り値を int 型にして下さい。HP が 0 になった時と ESC キーを押した時に真を返すようにして、main.c で真が返ってきた時にプログラムが終了するようにして下さい。

ヒント:ESC キーは VK_ESCAPE

最後に

Q&A

Q.画像の背景を透過するには？

A.マスク画像を使う方法、アルファチャンネル（透明度）を使う方法があります。

Q.爆発アニメーションを実現するには？

A.爆発のパラパラ漫画を作り、毎フレーム毎に転送する画像を変えていくことでアニメーションを実現できます。数フレームに 1 回切り替えるのであれば framecount 変数と%演算子を使いましょう。また、メンバーに x,y,frame を持つ EXPLOSION 構造体の配列を作ると良いでしょう。

```
BitBlt(hdc, exp[i].x, exp[i].y, bmpdc, IMAGE_SIZE * exp[i].frame, 96, SRCCOPY);
…
exp[i].frame++;
```

最後に

まだまだ、未完な部分が多いのですが、今回でソフトゼミ B は最後です。時間が経つと敵が発射する弾が増えるなど面白くするアイデアは無数にあるでしょう。時間があれば是非、さらなる改造を施してみて下さい。

また、「Windows ゲームプログラミング」は C 言語と Windows API でゲームを作る上で非常に有用な情報が載っているのでお勧めです。また次のサイトも 1 度見ておくと良いでしょう。

猫でもわかるプログラミング

http://www.kumei.ne.jp/

WisdomSoft

http://wisdom.sakura.ne.jp/

また、インターネット上を探してみると、色々な人が作った便利なライブラリ（関数群）があるので、必要に応じて利用すると良いでしょう。

参考文献

サンプルプログラムの作成の際に参考にした文献です。

Windows ゲームプログラミング

ソフトバンク パブリッシング 赤坂 玲音 著

基本的な Windows ゲームの開発方法について書かれています。C 言語の基礎を一通り理解している人向けの本です。

ゲームプログラミング Wiki

http://www.c3.club.kyutech.ac.jp/~sukiyaki/

九州工業大学のサークル C3(Composite Comuputer Club)の sukiyaki 氏のウェブサイトです。フレーム制御や当たり判定など、ゲーム開発における有用な情報が多数あり。

2006 年度 ソフトゼミ B テキスト

2006 年 03 月 07 日 Pre.1 発行

2006 年 03 月 17 日 Pre.2 発行

2006 年 06 月 29 日 完全版発行

2006 年 08 月 14 日 修正版発行

著作者 武山 文信

発　行 明治大学エレクトロニクス研究部

神奈川県川崎市多摩区東三田 1-1-1

明治大学生田校舎

部室センター 230

ウェブサイト URL

http://www.isc.meiji.ac.jp/~eleken/