WORD

2007.9 From College of Information Science

2

特集

書籍紹介

ニコニコ動画

ようこそ☆WorDonald's

シューティングゲーム開発講座

現代思想特講Ⅲ

消えたA棟の謎号

消えたA棟の謎

WORD

2007 年 8 月 26 日　13:50:00　執筆

もう少し楽しくなるニコニコ動画

この文章は 編集部 **Tait-you**さんが執筆した文章です

最近、様々な事で話題になっているニコニコ動画。みなさんの中にもアカウント登録してニコニコ楽しんでいる方も多いと思います。もちろん、ただテキトーに動画を見て、テキトーにタグを追加したり減らしたり、テキトーにコメントを書いているだけでも十分楽しいと思いますが、どうせなら、自分でも動画を作ってもっと楽しんでみてはいかがでしょうか。と、言うことで今回はニコニコ動画に投稿するための基本的な知識と技術を皆さんにも教えたいと思います。もちろん、*わざわざニコニコ動画の為にお金を使ったら負けかな*と思うみなさんの為に、可能な限り低予算での製作方法をお教えしたいと思います。

１・動画を作成するための基本的なツール

ニコニコ動画は flv 形式で再生をされています。しかし、投稿する場合は MPEG1、MPEG2、MPEG4、WMV、DIVX という形式でも可能とされているのですが、困ったことによく動画の再生とかに用いられる avi ファイル形式では投稿することは出来ません。そこで、ニコニコ動画を制作する手段には次の２つが考えられます。

・直接 flv や WMV 形式で動画を作成する。
・一旦他形式で作成した後、flv や WMV 形式で変換する。

基本的に前者の方が手間はかかりませんが、出来る動画には限界があります。せいぜい元が flash だったり、作業用 BGM のタグなので使われる程度だと思います。そこで、**基本的には後者での方法で制作をすることを前提に以下はお話ししていきます。**まぁ、どうせ使うファイルは前者は後者からいくつか抜いた程度ですしね。

まずは絶対に必要なソフトとして、**キャプチャソフト**と**動画編集ソフト**、それと**形式変更ソフト**を紹介します。

・キャプチャファイル編

a.カハマルカの瞳（http://www.paw.hi-ho.ne.jp/milbesos/）

フリーでありながらも非常に高機能とされているため、数多くの方が利用しているキャプチャソフトです。動画のキャプチャの他、音声のみのキャプチャや画像のみのキャプチャ、さらには動画の結合処理やカット処理も可能の優れものの為、これ一本あれば大抵の動画は録画できるようになります。

b.fraps 1.9d（http://www.gigafree.net/media/recording/fraps.html）

こちらは上では重くて綺麗に取れない動画（例えばゲームのプレイ動画など）を撮影するために使用するキャプチャソフトです。とにもかくにも動作が軽いので低スペックマシンでもゲーム動画をキャプチャできる実力を持っています。ただし、**音は別取り**です。また、一部ゲームではこのキャプチャマシン自体が起動しない場合もあります。ちなみにこの最新版は音も取ることも可能になりますし、細かい動画の設定も指定できますが、有料になります。そこまでしてでも高品質で作りたい場合はそちらをどうぞ。

c.キャプチャボード他色々一式（上級者向け）

SFC や PS でのゲーム動画を実機でやりたい人向け。当然ですがお金もかかりますし、玄人志向なのであまり初心者にはお勧めはしません。しかし、どうしてもこういうよなソフトを使ってでも超高画質で録画したいという場合はこちらのほうがいいでしょう。もちろん、アニメの場合も基本はこちらだと思います（ただし、アニメは著作権に引っかかること承知で**自己責任**でお願いします）。今回はここで説明してしまうと長くなってしまうのでそういう動画の作り方のサイトを紹介する事で割愛させて頂きます。

参考サイト　コンシューマーゲームのプレイ画面を録画する

http://www008.upp.so-net.ne.jp/groupsvs/column/c13.htm

・動画編集ソフト

a.Windows ムービーメーカー

（http://www.microsoft.com/japan/windowsxp/downloads/updates/moviemaker2.mspx）

簡単な動画を作るだけでしたら WindowsXP 標準装備のこれで十分事足ります。音楽タグでよく見かけるメドレー系ムービーは基本これで制作可能です。取った動画をほとんど加工しないのなら、ゲームプレイ動画もこれで多少の編集は可能です。

b.VideoStudio（http://www.ulead.co.jp/product/videostudio/function3.htm）

c.Adobe Premiere（http://www.adobe.com/jp/products/premiere/）

○○ P のタグを付けられた（or 自分でつけた）方は誰でも持っていると言われるご存じの高機能動画編集ソフト。動画に細かい加工をしたり複数の動画を重ねて使いたい場合などに必須…なのですが、大抵はこんな物使わなくとも何とかなります。ただ、動画にこれでもか！と言う加工をしたい場合は購入しても損は無いと思われます。自分の作りたい動画と相談してお使いください。

・形式変換ソフト

a.BitComet FLV Converte（http://cowscorpion.com/MultimediaTools/BitCometFLVConverter.html）

flv に動画を変換する場合はこちら。軽くて早くて扱いやすいのですが少々、画質が荒くなるのが難点かもしれません。ですが、基本はこれで問題はないと思います。フリーですし、使って損はないと思われます。

b.Windows Media エンコーダ

（http://www.microsoft.com/japan/windows/windowsmedia/9series/encoder/default.aspx）

エンコードに時間がかかってしまうのが最大の難点ですが、その代わりしっかりと高画質、高音質をサポートしてくれるソフトです。高画質。高音質を維持したい方向けかな？と思われます（ただし、時間帯によってはニコニコ内で再エンコードされてしまい、画質が落ちることがあるので注意です）。こちらも無料で使えます。

以上が基本的に必須と思われるソフトを色々ご紹介しましたが、これら全てを用意する必要はまったくありませんし、また、これ以外にも加工する内容によっては必要なソフトがあると思われます。**あくまでもこれらは基本的なソフトだと受け止めてもらうとありがたいです。**

２・動画を作成するときに覚えておきたいこと

次に動画を作る際に気をつけなければならない事をいくつか紹介したいと思います。重大な事としては、**一つのファイルにかけられる容量は flv 形式で 40MB、それ以外は 100MB までと言うことです。**画質の向上や音質の向上、そして見栄えが良くなるようにあれやこれやと加工すると気が付けば制限をすぎてしまったなんて事が多々あります。せっかく作ったのに制限オーバーで再加工しなければいけなくなった、なんて事にならないように。特にエンコードは時間がかかる事なので容量オーバーでエンコードしなおし、なんて事になるとまさに時間の無駄遣いなので絶対に忘れないように。やる気もなくなりますしね。

その代わりといっては何なのですが、一つのファイルに対する制限は厳しいですが、一つのアカウントに対する制限はかなり余裕があります。これはあくまでも私の場合なのですが、**大体、一つの動画で使用するファイル容量はせいぜい全体の 1%程度です。**つまるところ、単純計算すると、**一つのアカウントで 100 本もの動画ファイル登録できる**わけです（プレミアム会員ならばさらに倍）。よって長くなるな、と思った場合は素直に分割した方が吉と思われます。作業用 BGM とかでも無い限り、あまりにも長ったらしいファイルは嫌われますしね。

また、**自分のパソコンのスペックをしっかりと把握しておくことも大切**です。動画製作にはどの行程においてもかなりのパワーを必要とされます。少なくとも、そこらの PC ショップで売っている程度の物、特にノート PC では、動画一本ロクに作れない可能性も無きにしもあらずなのです。ゲーム動画とかアニメをエンコードしたいのなら、素直にデスクトップ PC を、しかも余裕があれば **CPU をデュアルコアにしたり、メモリを１ギガ**にしておきたいところです。

…とは言うものの、急に言われてもそんな PC 準備はできませんよね。ってか私の PC もノートでしかも大したスペック持ちではありませんし。そこで次の項ではそんな方でもすぐに作れるお手軽な動画の作成方法の一例を紹介したいと思います。

３・実際に動画を作ってみよう！

ではでは、早速動画を作ってみましょう。今回は誰でも比較的簡単に作れる作業用 BGM やメドレーを作っていきたいと思います。今回使用するのは項目 2 で紹介した、カハマルカの瞳、Windows ムービーメーカーを２つを基本とし、オプションとして、Windows Media エンコーダーと Audacity（http://www.forest.impress.co.jp/lib/pic/music/soundedit/audacity.html）というファイルを使用したいと思います。Audacity というソフトは様々な.wav ファイルの加工をするのに使われるソフトです。無音部分の作成や、ピッチ変更などもできて便利です。

step1　カハマルカの瞳を使って wav 形式で録音する。

最初の行程はカハマルカの瞳を使用して音を録音します。なぜ動画じゃないのか、と言いますと、動画で録画すると、スペックの低い PC では撮影した動画がカクカクになったり音ズレになったりするおそれがあるからです。よって、今回は音と画像は別取りをし、かつ、画像は静止画オンリーでいってみたいと思います。この方が綺麗ですし、制作もぐっと楽になります。

まずは録音環境の設定です。画面右下（にあると思われる）音量アイコンを右クリックし、のオーディオプロパティの調整をクリックし、オーディオ内の録音を音の再生と同じデバイスにしてください。これによってパソコン内での音楽がそのまま録音されることになります。

では、実際にカハマルカを起動してみましょう。起動したあとは以下の図を参考にしてボタンを調整した後、右下の REC ボタンを押せば、簡単に録音が始まります。終了させるときは、タスクトレイに現れた目のアイコンを右クリックし、停止を押せば OK です。

画像じゃ分かりづらいですが、WAVE の部分が赤くなっていれば OK

ちなみに*動画の場合*はもう少し手順は複雑ですが、一旦 TEST をクリックしてフレームレートを調整、**そこででたテスト結果のワンランク下の数値**で録画すればまず音ずれは無くなると思います（あまりにもフレームレートが低すぎてどう考えても動画がカクカクになると思う、あるいはなってしまった場合は、画質のビット数を落とすことでフレームレートを上げられる可能性があります。いずれにせよ、数値を変更したら念のため TEST を再使用することをオススメします)。ただ、これらはヘルプに詳しく書いてあるため、それを読めばまず困らないと思いますが…。

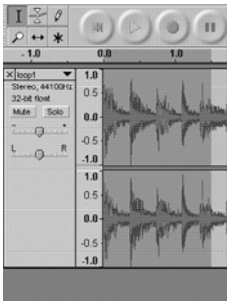
また、初めから MP3 等のファイルである場合はこれをする必要はありません。この動作が必要なのは MIDI や ogg など、WMP で再生できないファイルを使用したい場合の行為なので。

さて、録画・録音に成功したのはいいものの、余計なシーンや音が前後に入ってしまったという場合もあると思います。しかし、ご安心ください。録画の場合はカハマルカの瞳のカット編集で、録音の場合は先にあげた Audacity を利用すればこの余計な部分を消すことができます。今回は録音がメインですし、せっかく補助ツールとして紹介したのですから Audacity を使って前後の部分の削除に挑戦してみましょう。

Audacity を起動すると、作業ウィンドウが出てきますのでそこに加工したい音源（１曲入れるごとに別ウィンドウが開いてしまうので加工したいファイルを複数一度に入れることはオススメしません）をドラッグ＆ドロップしてください。すると波形が現れるはずです。この波形は２つで１セットになっており、上が左から聞こえる音、下が右から聞こえる音を波形として表したモノです。ちなみに音源左端にある、２つのスライドのうち、下

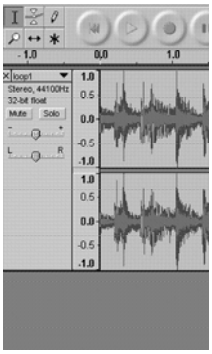
のL・Rがあるスライドを動かすことによってこのバランスを変更することができます(さらに余談ですが上のスライドは出力音量の調節に使います)。

さて、では余計な部分のそぎ落としをします。まずは、必要ない部分の波形をドラッグ&ドロップした後、編集をクリックするとメニュー中央部に２つのコマンドがあるはずです。それぞれを説明しますと、

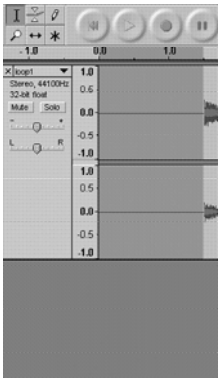


ちなみにこれが変更前

「削除」・・・選択した部分を削除します。選択した時間帯も削除されます。



「無音」・・・選択した部分を無音化します。選択した時間帯は削除されません。



つまり、上のは時間ごと、下は選択した時間は残したまま音だけを消すという違いがあります。で、どちらを使うか迷う方もいるかもしれませんが、ここは**削除をオススメ**します。と、いうのも無音部分自体は「製作」内の Silence で作成することができるし、また後のムービーメーカー内でも調整できますので余計な部分はそぎ落とした方が作業しやす

くなると考えられるからです。

wav ファイルの調整が終わったら、ファイルの別名で書き出し（もしくは MP3 での書き出し）を選択してファイル名をつけて保存します（元のファイルに上書きすると音がおかしくなってしまう時があるので、できれば**元のファイル名とは別に保存**をすることを強くオススメします)。

step2　ムービーメーカーで編集作業を行う。

さて、必要な音源と画像（こちらは各自用意）or 動画を準備したらムービーメーカーで編集を行います。ムービーメーカーを開いた後、先程の Audacity にファイルを入れたのと同じようにドラッグ＆ドロップします。画像と音源は同時にいれても一つのコレクション（動画を置いておくスペースの事）内に入るので一度に入れても構わないのですが、動画の場合は動画一つにつき、別のコレクションに置かれてしまいます。もちろん、画面左上のコレクションというボタンを押せば、全てのコレクションから自分の使いたいコレクションを後からでも選択できますが、ちょっと切り替えが面倒くさいですし、今回は無編集が前提なのでひとつづつ入れて、タイムラインに貼ってから次の動画を取り入れるという作業の方が良いと思います。

Windows ムービーメーカーには下部にタイムラインというのがあります。上から Video、切り替え効果、オーディオ/音楽の順に（設定によって他のタイムラインがあったり、順序が違う場合もあります）あるはずです。ちなみに下にストーリーボードがある場合は、タイムラインを表示してください。今回の場合はこちらで作業する方が楽なので。

あとは基本は同じです。**使いたい場面に画像や音楽をドラッグ＆ドロップをすれば簡単に作成できる**でしょう。ね？簡単でしょう…と言ってしまえばお終いなのですがそれではあまりに手抜きすぎるので少し補足を。

静止画を出現させている時間は初期設定では５秒間ですが、画像の右端をドラッグ＆ドロップすることで時間を伸ばしたり縮めたりする事が出来ます。ただし、この操作による時間の変化は**驚くほど大雑把**です。基本的には下のオーディオの切れ端を自動的にサーチしてそこに移動しようとします。画像とオーディオで変化を一致させたいときは便利ですがそうでないときは**出来る限り編集したい部分を拡大して操作をすると細かい時間帯を設定しやすくなります。**

また、画像（ムービー）は途中で隙間を空けることはできませんが、オーディオは途中で無音時間を作ることが可能です。よって少しこの音楽とこの音楽に合間をつけたいなと思うときは移動させたい音を例のごとくドラッグ＆ドロップしましょう。これも先程のと同様、大雑把＋自動的に上の画像と一致させようとしますので細かく設定したい場合は可能な限りタイムラインを拡大させて行ってください。

さらに動画の音とオーディオの音は合成させることができます。ただ、これは逆に言えば、元の動画に元動画と違った別の音を入れようとする場合は動画の方の音源を消す必要があるという事になります。この処理はここではなく、**先程のカハマルカの瞳のカット編集で加工可能**なのでそちらで行ってください。

さて、完成したらムービーを出力します。ムービー編集にあるコンピューターに保存を選んでください。後は画面の指示に従って内容を書き込めばあとは自動で出力されます。ただし、長さによってはかなり PC に負担のかかる作業になりますので待っている間に同じパソコンでニコ動なんて見てないように。完成するまで家事をするなりなんなりで時間をつぶしましょう。

最後に動画を完成させたらその動画の大きさを確認しましょう。100MB を越えていない場合はこのまま完成となります。大体、20 ～ 30 分くらいの動画なら十分範囲内に収まると思います。もし 100MB を越えていたら Windows エンコーダーで画質や音質を１ランク落としてみましょう。それでも 100MB を越えるようであれば、２つに分割するなどして、工夫してみましょう（画質、音質共に 2 ランク以上落とすとかなり質が悪くなるのでやらないことをおすすめします)。

step3　ニコ動にうｐしてみる。＋うｐしたその後の注意点

データが完成したら SMILE　VIDEO から作品をアップロードしましょう。こちらも画面の流れに沿っていけば私の説明なんて入らないような気がしますので…、何点か補足をしておきましょう。

まず最初に気をつけたいのが**アスペクト比**です。…とはいうものの、基本的に今回で作った作品は気にせず 4:3 一択ですが DVD から取ってきたとかハイビジョン放送を録画したという場合は 4:3 では縦長、あるいは横長の変な映像になってしまうので、そう言う場合は 16:9 をオススメします。これは一度作品をアップデートさせてしまったらどうしようもないため、十分に気をつけてください。

先のアスペクト比程重要ではありませんが、それでも優先的に力を入れたい一つに**サムネイル**があります。サムネイルは言わば店の一押し部分を見せるところです。特に明確なシリーズではないとかではない限り、サムネイルは皆がみたいと思われる画像を用意することをオススメします。サムネにつられて・・・系統のコメントを貰えるサムネイルをうまく取れたら上出来かな、と思います。昔はランダムでしか作れませんでしたが、今では時間指定もできるようになったため、自分の思い通りのサムネイルを作ることも難しいことはないでしょう。

最後にうｐした後のあなたの取るべき行動です。うｐしたから、ハイ終わり…では人だってやってきません。少なくとも２週間くらいは定期的にコメントの確認、タグの確認、それに応じた説明文の細かな変更をすると好感度は上がります。シリーズ物の場合はシリーズ共通タグをロックしたり、誘導用のマイリストや sm を説明文に書くなどするのも忘れずに。余力があるのなら、下のニコニコ市場に色々追加するのもありかもしれません。

逆に、才能の無駄使い、もっと評価されるべき、吹いたら負けのカテゴリを自分でつけてロックをかける、必要以上に自分の動画に自分のコメントを乗せて定期的に挙げる、必要以上に自分の作品をマイリストに（複数アカを使って）登録するのは**いくら素晴らしい動画でも荒らされる対象になってしまう**（ランク１位が取れないなど、演出上の場合は別ですが）ので、絶対に行わないことを心掛けてください。もちろん、空気を読まない発言を説明文に書くのもいけませんよー。投げっぱなしジャーマンよりは定期的に説明文を調整したりするのは好感度を挙げるコツです。忘れないように！

４・ニコニコ動画に動画をあげる現状とは？

最後に、ニコニコ動画に対するいくつかの現状をお伝えしたいと思います。まずは、再生数とコメント数ですが、良作動画と認められた、と思って良い目安はアンケートを除き、**再生数 1 万、1000 コメ**となります。アンケートのみ、その動画の性質より、**再生数 10 万、1 万コメ**が基本的な良作と思って問題はないでしょう。もちろん、これは数値的な意味であり、時期や隠れた名作などもあるため、一概にこれが全てだ！と言っている訳ではありません。ただ、大体１ヶ月くらい経っている動画ならある程度参考にしてよい数字だと思い

ます。と、言うのも、アンケートのカテゴリタグの動画を除くと、**どちらの動画もほとんど全体の１割あるかないか、と言われるようなのが現状**なのです。極端な話ですが、*基本的にニコニコ動画でこれは面白いと言われている作品はこの１割の中から出ていると言えます*。では、この再生数 1 万、1000 コメを達成するにはどんな事をすればいいのかを考えてみましょう。

実のところ、高画質・高音質は確かに動画の再生数、コメント数を挙げるのには確かに有効な手段の一つですが、主要素ではないと思います。現にかなりの低画質、低音質でも 1 万、1000 コメを超えている動画は決して少なくはありません。では、何が一体重要なのか？答えは大きく分けて 2 つあります。

1 つは**『独創的な作品を作る』**と言うことです。いくら高音質、高画質でも、人と似たような作品では見てくれる方はそう多くはありません。それよりも、他の人とは明らかに違った作品に対しては非常に食いつきは良いモノになります。○○の無駄遣いというタグがいくらかあるのは、これはつまり、そのお馬鹿な事柄に全力で挑む姿勢に心打たれた（腹筋が崩壊した？）人に与える賞賛のようなものです。人とは違う、それだけでもかなり動画としての求心性は高まるモノであると断言できます。

ただ、いくら独創的な作品でも、見て貰えない場合があります。それは何故か？アイドルマスターの高木社長ではありませんが、それは日々の流行情報を見逃している、のが原因です。もう分かりましたね？もう 1 つの見てくれるための努力、それは**『需要に合わせる』**です。もちろん、全部が全部流行に合わせろ、とは言いません。例えば、何か歌うにせよ、オリコン 1 位の曲を歌っても、ニコニコ動画には通用しません（…ただし、発想を転換して、淡々と毎週のオリコン 1 位の曲を歌ってみればあるいは新しい風を起こせるかもしれませんが…）。むしろ、『創聖のアクエリオン』や『エアーマンが倒せない』を歌った方が見てくれる人は多いでしょう。ようはそう言うことです。自分がやりたい何かの中にもそっと、その見せる場所の流行を取り入れておくだけで結構差というモノはあると思います。

…と、ここまで言ったところで全てを無に帰すような発言を致しますが、**結局の所、最終的に何がヒットするかは、実際に作品をうｐするまで分からない事がほとんどです。**何気なく面白そうだなと言う安易な発想で作った作品が案外ものすごい評判になることも少なくはありません（…ってか、現に私はそんな発想で先の再生数 1 万、1000 コメントをあっさりと超えてしまった一人だったりします）。**大切なのは、いかに優れた作品を作るよりもいかにみんなでそれこそ、ニコニコできるような作品を作ることが大切なのでは無いでしょうか？**まぁ、最初からいきなりヒットする作品を作ろうといきごむことはせず、のんびり、まったり、面白そうな作品を淡々と作っていけば、もう少しだけ、ニコニコ動画が楽しくなるかもしれませんよ…とタイトルに戻ったところで、今回の話は終わりにしたいと思います。さぁ、あなたもニコニコ動画うｐ主の世界にレッツゴー！

シューティングゲーム開発講座

文 編集部 yasuharu

目次

今回は、第二回ということで、5 章、6 章の部分を説明します。
（太字の部分が今回の記事です）

（打ち消し線があるところは、変更された部分です。
また、7 章以降は予定です）

前回の説明については、ネット上に掲載していますので、そちらの方を参照してください。

シューティングゲーム開発講座 第 1 回
http://yasuharu.net/word/shooting/1/

また、説明のために前回までのコードをここに書きます。後からの説明で使用します。

```
1.      #include <windows.h>
2.
3.      #define CLASS_NAME "Sample"
4.      #define TITLE_NAME "Shooting Game"
5.
6.      typedef struct WindowInfo
7.      {
8.              int Height;
9.              int Width;
10.     } WindowInfo;
```

```
11.
12.     typedef struct Player
13.     {
14.             POINT Position;
15.             int Width;
16.             int Height;
17.             int HitPoint;
18.             HDC hImage;
19.             WindowInfo mWindowInfo;
20.     } Player;
21.
22.     HINSTANCE ghInst;
23.
24.     LRESULT CALLBACK WindowProc(HWND hWnd,UINT msg,WPARAM wp,LPARAM lp);
25.
26.     int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrevInst,LPSTR lpsCmdLine,int
nCmdShow)
27.     {
28.             MSG msg;
29.             BOOL bRet;
30.             HWND hWnd;
31.             WNDCLASSEX WindowClass;
32.
33.             ghInst = hInst;
34.
35.             WindowClass.cbSize = sizeof(WNDCLASSEX);
36.             WindowClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
37.             WindowClass.lpfnWndProc = WindowProc;
38.             WindowClass.cbClsExtra = 0;
39.             WindowClass.cbWndExtra = 0;
40.             WindowClass.hInstance = ghInst;
41.             WindowClass.hIcon = (HICON)LoadImage
42.             (
43.                     NULL,MAKEINTRESOURCE(IDI_APPLICATION),
44.                     IMAGE_ICON,0,0,
45.                     LR_DEFAULTSIZE | LR_SHARED
46.             );
47.             WindowClass.hCursor = (HCURSOR)LoadImage
48.             (
49.                     NULL,MAKEINTRESOURCE(IDC_ARROW),
50.                     IMAGE_CURSOR,0,0,
51.                     LR_DEFAULTSIZE | LR_SHARED
52.             );
53.             WindowClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
54.             WindowClass.lpszMenuName = NULL;
55.             WindowClass.lpszClassName = CLASS_NAME;
56.             WindowClass.hIconSm = (HICON)LoadImage
57.             (
58.                     NULL,MAKEINTRESOURCE(IDI_APPLICATION),
59.                     IMAGE_ICON,0,0,
60.                     LR_DEFAULTSIZE | LR_SHARED
61.             );
62.
```

```
63.             RegisterClassEx(&WindowClass);
64.
65.             hWnd = CreateWindow
66.             (
67.                     CLASS_NAME,TITLE_NAME,
68.                     WS_OVERLAPPED|WS_SYSMENU,
69.                     CW_USEDEFAULT,CW_USEDEFAULT,
70.                     640,480,
71.                     NULL,NULL,ghInst,NULL
72.             );
73.
74.             ShowWindow(hWnd,SW_SHOWNORMAL);
75.
76.             while((bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
77.             {
78.                     if (bRet == -1)
79.                     {
80.                             break;
81.                     }else{
82.                             TranslateMessage(&msg);
83.                             DispatchMessage(&msg);
84.                     }
85.             }
86.
87.             return 0;
88.     }
89.
90.     LRESULT CALLBACK WindowProc(HWND hWnd,UINT msg,WPARAM wp,LPARAM lp)
91.     {
92.             PAINTSTRUCT ps;
93.             HDC hdc;
94.             RECT rect;
95.             static Player mPlayer;
96.             static WindowInfo mWindowInfo;
97.
98.             switch(msg)
99.             {
100.            case WM_CREATE:
101.                    GetClientRect(hWnd,&rect);
102.                    mWindowInfo.Width = rect.right;
103.                    mWindowInfo.Height = rect.bottom;
104.
105.                    mPlayer.mWindowInfo = mWindowInfo;
106.                    mPlayer.Position.x = 200;
107.                    mPlayer.Position.y = 400;
108.                    mPlayer.Width = 16;
109.                    mPlayer.Height = 16;
110.                    mPlayer.HitPoint = 10;
111.
112.                    break;
113.            case WM_PAINT:
114.                    hdc = BeginPaint(hWnd,&ps);
115.                    TextOut(hdc,mPlayer.Position.x,mPlayer.Position.y,"■",2);
```

```
116.                    EndPaint(hWnd,&ps);
117.
118.                    break;
119.            case WM_CLOSE:
120.                    DestroyWindow(hWnd);
121.                    break;
122.            case WM_DESTROY:
123.                    PostQuitMessage(0);
124.                    break;
125.            }
126.
127.            return DefWindowProc(hWnd,msg,wp,lp);
128.    }
```

5. プレイヤー機体の移動

1. タイマーの設定

この章では、プレイヤーの機体の移動について説明をしていきます。プレイヤーの機体の移動は、前回の説明にあった描画部と操作部の 2 つのうちの後者の部分にあたります。操作部では、ユーザーからのキー操作を受けて、プレイヤーの機体の位置を決めます。そして、その位置情報を変数に入れておきます。ちなみに、描画部というのは、この位置情報を入れた変数をつかって、プレイヤーの機体の描画を行います。これは、前回説明した部分です。

まずは、ユーザからのキー操作をどのようにして取得するのか、説明していきます。ユーザが押したキーを取得する方法としては、主に次の二つの方法があります。

一つは、WM_CHAR ウィンドウメッセージなどを使って取得する方法です。WM_CHAR ウィンドウメッセージというのは、ユーザがキーを押したときにウィンドウプロシージャに送られてくるウィンドウメッセージです。このメッセージが送られてきたら、どのキーが押されているか判断して、その後の処理を行います。

もう一つの方法は、Win32API のタイマー機能と GetKeyState 関数を使う方法です。今回は、こちらの方法を使って説明をしていきます。

まず、「タイマー機能ってどんなのだろう？」と思うかと思います。簡単に言ってしまえば、普段食事を作るときなどに使っているタイマーと似たようなものです。Win32API のタイマー機能では、タイマーを設定すると Windows が指定した時間ごとに特定のウィンドウメッセージを送ってくれます。

では、細かくみていきましょう。まず、タイマーを作成します。タイマーの作成は SetTimer 関数を使って行います。SetTimer 関数の引数などは、次のようになっています。

```
    UINT_PTR SetTimer(
            HWND hWnd,              // ウィンドウのハンドル
            UINT_PTR nIDEvent,      // タイマの識別子
            UINT uElapse,           // タイムアウト値
            TIMERPROC lpTimerFunc   // タイマのプロシージャ
    );
```

（引用元）

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpwinui/html/_win32_settimer.asp

上から説明していきます。ウィンドウハンドルは、メッセージを受け取るウィンドウのハ

ンドルを指定します。識別子は、タイマーを識別するための番号です。識別子を使って、複数のタイマーが存在するときでも、どのタイマーのメッセージなのか見分けられるようにします。タイムアウト値は、メッセージが送られてくるまでの時間を指定します。単位はミリセカンドです。最後に、タイマーのウィンドウプロシージャは、今回は関係ないので NULL にしておきます。

SetTimer 関数を使うとタイマーが作成されます。タイマーが作成されると、ウィンドウプロシージャに対して、指定したタイムアウト値の間隔で WM_TIMER メッセージが送られてきます。このとき、ウィンドウプロシージャの引数の wp のなかに、タイマーの識別子が入っています。SetTimer 関数を使うと、すぐにタイマーのカウントが始ります。この処理の仕方については、後から実際にコードを示して解説します。

最後に、タイマーを使い終わったら破棄するようにします。破棄をするには、KillTimer 関数を使います。この関数の引数などは次のようになっています。

```
BOOL KillTimer(
        HWND hWnd,              // ウィンドウのハンドル
        UINT_PTR uIDEvent       // タイマの識別子
);
```

（引用元）

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpwinui/html/_win32_killtimer.asp

ウィンドウのハンドルは、破棄したいタイマーがあるウィンドウのハンドル（つまり、作成したときと同じウィンドウハンドル）を指定します。タイマーの識別子も作成したときと同じものを使います。

以上が、タイマーの作成と破棄についての説明です。それでは、実際のコードをみていきましょう。次のように、コードを追加してください。

コード 1（P.23 4 行目の後に追加）

```
#define TIMER_ID 10000
```

コード 2（P.25 110 行目の後に追加）

```
        SetTimer(hWnd,TIMER_ID,1000 / 60,NULL);
```

コード 3（P.26 118 行目の後に追加）

```
    case WM_TIMER:
        if((int)wp == TIMER_ID)
        {
        }
        break;
```

コード 4（P.26 119 行目の後に追加）

```
    KillTimer(hWnd,TIMER_ID);
```

まず、SetTimer 関数のところ（コード 2）をみてください。TIMER_ID はコードが見やすくなるように、コード 1 で define しています。また、タイムアウト値として 1 秒間に 60 回メッセージを送るようにしています（コード 3）。

次に、case WM_TIMER で始まる一連の処理をみてください。タイマーの識別子は WindowProc 関数の wp 引数に格納されているので、wp と TIMER_ID を比較しています。タ

イマーの中での処理については後から説明をします。

2. キーの取得

次に、キーの状態の取得を行っていきます。キーの取得には GetKeyState 関数を使います。この関数では、キーが押されているかどうかなどの状態を取得します。GetKeyState 関数は次のようになっています。

```
SHORT GetKeyState(
        int nVirtKey        // 仮想キーコード
);
```

（引用元）

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpwinui/html/_win32_getkeystate.asp

ここで出てくる仮想キーコードというのは、キーごとに割り振られた定数のことです。この定数については、詳しくはこちらに載っています。

仮想キーコード
http://yokohama.cool.ne.jp/chokuto/urawaza/prm/virtual_key_code.html

今回使用するものだけこちらに抜粋しました。

VK_LEFT	矢印キーの←	VK_UP	矢印キーの↑
VK_RIGHT	矢印キーの→	VK_DOWN	矢印キーの↓

GetKeyState 関数の引数に、この仮想キーコードを渡してキーの状態が返ってきます。戻り値は、2 進数で表したときに、最上位ビットが 1 ならキーが押されていて、0 なら押されていないことになります。つまり、戻り値が負であれば押されていて、正であれば押されていないことになります。

さて、これでキーが押されたかどうか確認する一連の方法について説明が終わりました。これらをもとに実際に実装をしていきます。大まかな流れとして、プレイヤーを移動させる関数、プレイヤーの移動処理を行うときに呼び出される関数の二つを作成します。そして、これらの関数を呼び出すためのコードを書きます。では、コードを見ていきましょう。

コード 5（P.23 1 行目の後に追加）

```
#define _USE_MATH_DEFINES
#include <math.h>
```

コード 6（P.23 4 行目の後に追加）

```
#define MOVE_POINT 5
#define MOVE_POINT_NANAME（MOVE_POINT * M_SQRT1_2）
```

コード 7（P.25 88 行目の後に追加）

```
void SetXToPlayer(Player *mPlayer,int value)
{
        if(0 <= value && value < mPlayer->mWindowInfo.Width - mPlayer->Width)
        {
                mPlayer->Position.x = value;
        }
```

```
}

void SetYToPlayer(Player *mPlayer,int value)
{
	if(0 <= value && value < mPlayer->mWindowInfo.Height - mPlayer->Height)
	{
		mPlayer->Position.y = value;
	}
}

void PlayerMove(Player *mPlayer)
{
	int Key_Left = 0,Key_Right = 0,Key_Up = 0,Key_Down = 0;

	Key_Left = GetKeyState(VK_LEFT);
	Key_Right = GetKeyState(VK_RIGHT);
	Key_Up = GetKeyState(VK_UP);
	Key_Down = GetKeyState(VK_DOWN);

	if(Key_Down < 0 && Key_Right < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x + MOVE_POINT_NANAME);
		SetYToPlayer(mPlayer,mPlayer->Position.y + MOVE_POINT_NANAME);
		return;
	}
	if(Key_Down < 0 && Key_Left < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x - MOVE_POINT_NANAME);
		SetYToPlayer(mPlayer,mPlayer->Position.y + MOVE_POINT_NANAME);
		return;
	}
	if(Key_Up < 0 && Key_Right < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x + MOVE_POINT_NANAME);
		SetYToPlayer(mPlayer,mPlayer->Position.y - MOVE_POINT_NANAME);
		return;
	}
	if(Key_Up < 0 && Key_Left < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x - MOVE_POINT_NANAME);
		SetYToPlayer(mPlayer,mPlayer->Position.y - MOVE_POINT_NANAME);
		return;
	}

	if(Key_Left < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x - MOVE_POINT);
		return;
	}
	if(Key_Right < 0)
	{
		SetXToPlayer(mPlayer,mPlayer->Position.x + MOVE_POINT);
		return;
```

```
        }
        if(Key_Up < 0)
        {
                SetYToPlayer(mPlayer,mPlayer->Position.y - MOVE_POINT);
                return;
        }
        if(Key_Down < 0)
        {
                SetYToPlayer(mPlayer,mPlayer->Position.y + MOVE_POINT);
                return;
        }
}
```

```
        case WM_TIMER:
                if((int)wp == TIMER_ID)
                {
```

コード 8（P.27 コード 3 の if((int)wp == TIMER_ID)の 2 行後に追加）

```
                        PlayerMove(&mPlayer);
                        InvalidateRect(hWnd,NULL,TRUE);
                }
                break;
```

では、上から説明していきます。まず、一番最初の include しているところ（コード 5）の説明です。ここでは、math.h という数学関連の関数が入っているヘッダファイルをインクルードします。その時に、ヘッダファイルをインクルードする前の行で「_USE_MATH_DEFINES」としています。これは、math.h の中にあるπや自然対数の底の e などの定数を使用するときに define します。

次に、SetXToPlayer 関数、SetYToPlayer 関数について、説明をします（コード 7）。これらの関数では、プレイヤー構造体のポインタと X 軸・Y 軸の成分を引数とします。ここで、前回定義したプレイヤー構造体がどのようになっていたか取り上げると、次のようになっていました。

```
typedef struct Player
{
        POINT Position;
        int Width;
        int Height;
        int HitPoint;
        HDC hImage;
        WindowInfo mWindowInfo;
} Player;
```

SetXToPlayer 関数、SetYToPlayer 関数の引数がプレイヤー構造体のポインタとなっているのは、引数として与えた変数自体を書き換えなければならないためです。ポインタ渡しでない場合、それは変数のコピーを渡すことになってしまいます。すると、関数の呼び出し元の変数の値は変わりません。

SetXToPlayer 関数、SetYToPlayer 関数についての説明に戻ります。各関数では、それぞれ

の成分をプレイヤー構造体のポインタに設定します。このとき、この関数では、プレイヤーが画面外に出てしまわないように、範囲チェックを行います。

たったこれだけのコードだったら「別に関数に分けなくてもいいんじゃないか？」と思うかもしれません。ですが、この先コードの量が増えていくにつれて、この関数の中身と同じコードが増えていきます。同じコードが増えてしまうと冗長になってしまいます。まだ、冗長になるだけならいいかもしれません。ですが、もし、この関数の中身のコードに不具合があったとしたらどうなるでしょうか。もちろん、その不具合を修正することになるのですが、関数としてまとめていない場合、何カ所も変更を加えなければなりません。数カ所程度なら、まだ何とかなります。ですが、数百カ所となったらものすごく膨大な量です。そんなときに、関数にまとめてあったら、そこを書き換えるだけでいいわけです。ですから、同じコードがでてきそうだと思ったら、関数としてまとめてしまいましょう。

次に、PlayerMove 関数の説明です。この関数で押されているキーのチェックと、それに対する移動を実装しています。最初に斜め方向のキーが押されているかチェックします。押されていたら、現在の座標に移動量を加算し、SetXToPlayer（SetYToPlayer）関数を使って、座標を設定します。

また、斜め方向の移動を行うときは、単純に X 成分・Y 成分に MOVE_POINT を加算してしまうと、斜め方向の移動だけ移動速度が速くなってしまいます。そこで、斜めの方向が MOVE_POINT になるように計算しています。斜め方向の移動を MOVE_POINT にするためには、X 成分・Y 成分をそれぞれ $1 / \sqrt{2}$ 倍します。このとき、先にインクルードした math.h の中に M_SQRT1_2 という定数があるので、これを使用します。この定数は、$1 / \sqrt{2}$ として定義されています。これらを使って、斜め移動の時の各軸への移動量を MOVE_POINT_S として定義しています（コード 6）。

移動の処理を実装したら、PlayerMove 関数を呼び出すようにしましょう。この関数は、タイマーが呼び出された時に呼び出します。

最後に、プレイヤーの移動が終わったら、画面の表示の更新を行います（コード 8）。画面の更新には、InvalidateRect 関数を使います。InvalidateRect 関数は次のようになっています。

```
BOOL InvalidateRect(
        HWND hWnd,              // ウィンドウのハンドル
        CONST RECT *lpRect,     // 長方形の座標
        BOOL bErase             // 消去するかどうかの状態
);
```

（引用元）

http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/jpgdi/html/_win32_invalidaterect.asp

この関数では指定した長方形の座標を再描画します。ウィンドウハンドルには、再描画の対象となるウィンドウのハンドルを指定します。長方形の座標には、再描画する長方形の座標を RECT 構造体を使って渡します。このとき、長方形の座標に NULL を渡すことで、画面全体を再描画の対象にします。消去するかどうかの状態は、再描画を行う場所を削除するかどうかを指定します。

ここでは、全画面を一端削除して再描画を行っています。さて、これで、プレイヤーの機

体の移動が実装できました。試しに、コンパイルして、実行してみましょう。F5 キーを押せば、コンパイルから実行までやってくれます。正常に動作をしていれば、キーを押すとプレイヤーの機体が動きます。画面にちらつきがあるかもしれませんが、それについては、後の章で説明をします。

6. 敵の機体の表示

1. 敵のチェインを作る

この章では敵の作成を行っていきます。敵の作成は、大まかにはプレイヤーの作成と同じです。ですが、敵はプレイヤーと違って破壊されたり生成されたりを繰り返します。また、プレイヤーは画面上に一つですが、敵は画面上に何体もいます。

そこで、敵を破壊したり作成したりするのに、メモリの動的確保という方法を使います。メモリの動的確保というのは、任意のタイミングでメモリ（変数）を生成したり、削除したりします。コンパイル時にどれだけのメモリが必要になるか、わからないときにこの方法を使います。

また、複数の敵を管理するのには配列を使って管理するようにします。動的に確保したメモリの配列を使って管理します。ここでは、敵のチェインという表現を使っていますが、敵の連なった配列のことをこう呼んでいます。

それでは、敵のチェインを作成していきましょう。まずは敵の構造体の宣言です。

コード 9（P.24 20 行目の後に追加）

```
typedef struct Enemy
{
        POINT Position;
        WindowInfo mWindowInfo;
        int Width;
        int Height;
        int Destroy;
        HDC hImage;
        double Speed;
        Player *mPlayer;
} Enemy;
```

いくつかのものはプレイヤーの構造体と同じです。プレイヤーの構造体と違うところだけ説明をします。

Destroy は、機体が破壊されたかどうかを示す値です。この変数の値が 1 だった場合、機体が破壊されていることを示します。どうしてこうする必要があるかは、後から説明をします。

Speed は、敵の移動スピードです。プレイヤーの場合、MOVE_POINT という定数で指定していました。敵の場合、それぞれの敵で速度を変えるために設定できるようにしています。

mPlayer は、プレイヤー構造体変数へのポインタです。敵がプレイヤーに対して攻撃をするときなどに使います。

次に、敵へのポインタの配列を作成します。

コード 10（P.23 4 行目の後に追加）

```
#define MAX_ENEMY 100
```

コード 11（P.25 96 行目の後に追加）

```
static Enemy *mEnemy[MAX_ENEMY];
```

この配列の中に、動的に確保したメモリを入れます。

2. 敵の作成

次は、作成したチェインの中に敵を作成していきます。作成するためにメモリの動的確保を使用します。メモリの動的確保には、malloc 関数と free 関数を使います。それぞれの関数は次のようになっています。

```
void *malloc(
        size_t size
);
```

（引用元）*http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/vclib/html/_CRT_malloc.asp*

```
void free(
        void *memblock
);
```

（引用元）*http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/vclib/html/_CRT_free.asp*

malloc 関数は、メモリを確保するために使います。引数として確保するメモリのサイズを指定します。戻り値に確保したメモリのポインタが返ってきます。

free 関数は、malloc 関数で確保したメモリを削除するのに使います。malloc 関数で確保したメモリは確実に free 関数で削除するようにしてください。削除をしないと、メモリリーク*1 を起こしてしまい、Windows が不安定になる原因となります。

これらの関数を使って、敵の作成を行っていきます。まず最初に、敵を作成する関数と削除する関数の二つを作成します。敵を作成する関数では malloc 関数をつかってメモリの割り当てを行って、敵のパラメータを設定します。敵を削除する関数では、free 関数を使ってメモリを削除します。では、コードを見てみましょう。

コード 12（P.25 88 行目の後に追加）

```
Enemy* CreateEnemy(POINT Position,double Speed,WindowInfo mWindowInfo,Player *mPlayer)
{
        Enemy *mEnemy;

        mEnemy = (Enemy *)(malloc(sizeof(Enemy)));
        mEnemy->Destroy = 0;
        mEnemy->Position = Position;
        mEnemy->mWindowInfo = mWindowInfo;
        mEnemy->Height = 16;
```

*1 メモリリーク　確保したメモリの解放を忘れ、確保したままになってしまうこと。

```
        mEnemy->Width = 16;
        mEnemy->mPlayer = mPlayer;
        mEnemy->Speed = Speed;

        return mEnemy;
}

void DestroyEnemy(Enemy *mEnemy)
{
        free(mEnemy);
}
```

まず、CreateEnemy 関数では Enemy 構造体のメモリを確保しています。malloc 関数の引数のメモリサイズは、sizeof 演算子で求めています。sizeof 演算子では変数の型を引数として、その変数に必要なメモリのサイズを取得することができます。メモリを確保したら、変数に初期値を設定し、確保したメモリのアドレスを返すようにしています。

DestroyEnemy 関数では、free 関数を呼び出してメモリの解放を行っています。CreateEnemy 関数との関係がわかりやすくするために、この関数を使って free をするようにしています。

次に、今作成した関数を使用して、敵を自動的に作成するようにします。方法としては、まず、前の章で作成したタイマーを利用します。タイマーが呼び出されたときに乱数を使って、一定の確率で敵を作成するようにします。乱数については後から説明をします。そして、先ほど作成した CreateEnemy 関数を使ってチェインの空きの場所に敵を作成します。

乱数の作成について説明をします。乱数を作成するには、最初に乱数の種となる数を設定します。乱数の種を設定するには srand 関数を使います。この関数は次のようになっています。

```
        void srand(
                unsigned int seed
        );
```

（引用元）*http://msdn2.microsoft.com/ja-jp/library/f0d4wb4t(VS.80).aspx*

この乱数の種によって乱数の出方が変わります。また、種の値が同じの場合、乱数の出方は同じになります。そこで、一般的にランダムな値を取りたい場合は、srand 関数の引数に現在の時間を渡して乱数の種にします。そうすることで、いつもランダムな値が出るようになります。現在の時間を取得するには time 関数を使います。time 関数は次のようになっています。

```
        time_t time(
                time_t *timer
        );
```

（引用元）*http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/vclib/html/_crt_time.asp*

引数に時間を格納する変数のポインタを渡します。今回は NULL を渡していますが、こうすることでも時間が取得できます。次に、乱数を取得するには rand 関数を使います。rand 関

数は次のようになっています。

```
int rand( void );
```

（引用元）*http://msdn2.microsoft.com/ja-jp/library/398ax69y(VS.80).aspx*

rand 関数の戻り値として、ランダムな値が返ってきます。返ってくる値は、0 ～ RAND_MAX（Visual Studio の場合、RAND_MAX の値は 32767 となる）の範囲の整数が返ってきます。

では、コードを見ていきましょう。

コード 13（P.23 1 行目の後に追加）

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

コード 14（P.23 4 行目の後に追加）

```
#define ENEMY_SPEED 4
```

コード 15（P.25 96 行目の後に追加）

```
int i;
```

コード 16（P.25 110 行目の後に追加）

```
for(i = 0 ; i < MAX_ENEMY ; i++)
{
        mEnemy[i] = NULL;
}

srand((unsigned int)time(NULL));
PlayerMove(&mPlayer);
```

コード 17（P.30 コード 8 PlayerMove 関数の後に追加）

```
if((rand() % 60) == 0)
{
        for(i = 0 ; i < MAX_ENEMY ; i++)
        {
                if(mEnemy[i] == NULL)
                {
                        POINT Position;

                        Position.y = rand() % mWindowInfo.Height;
                        Position.x = rand() % mWindowInfo.Width;

                        mEnemy[i] = CreateEnemy(
                          Position,ENEMY_SPEED,mWindowInfo,&mPlayer);
                        break;
                }
        }
}
InvalidateRect(hWnd,NULL,TRUE);
```

まず、コード 16 の部分について説明をします。ここでは、チェインの中に NULL を入れています。最初にすべての要素の中に NULL を入れておくことで、どの要素が空きかわかるようにします。そして、srand 関数を使って乱数の種を生成しています。

次に、コード 17 の部分です。ここでは一定の確率で敵の生成を行っています。

一般に、乱数を使って一定の確率を表現したい場合、余りをとって比較を行います。どういうことかというと、rand 関数で返ってきた値を x、割る値を n とします。このとき、x / n の余りは 0 から n - 1 までです。そうすると、rand 関数で返ってくる値が十分大きいとき、余りの数はどれも同じ確率となります。今回の場合、rand 関数を 60 で割っています。そうすると、余りは 0 から 59 のどれかになります。そのうちの 0 になるときに敵を生成するようにしているので 60 分の 1 の確率で生成しているのと同じこととなります。

同じようにして、乱数を一定の範囲内におさめたい場合も余りを使って表現します。これは、Position.x、Position.y を設定している部分で使っています。

敵を生成するときには、まず、チェインの中を調べます。このとき、NULL であればその要素には敵が作成されていないので for ループで該当する要素を探します。NULL である要素が見つかったら敵の初期位置を決めます。NULL である要素が見つからなかったら、敵の数が最大であるので、それ以上敵を作成しないようになっています。敵の初期位置は、上で説明したように、余りを使って画面内におさまるようにします。そして、CreateEnemy 関数を使って敵を作成します。ENEMIY_SPEED は今のところは定数にしておきます。作成が終わったら break をして、for ループから抜けます。以上で敵の作成は終わりです。

作成の部分と対になるようにして、削除する部分を説明していきます。敵を削除するタイミングには 2 つあります。一つは、敵の Destroy フラグが 1 の時です。これは、タイマーが呼び出されたときにチェックをします。もう一つは、ゲーム終了時です。ゲームが終わるときにも、しっかりとメモリの解放を行うようにしましょう。では、コードを見てみましょう。

コード 18（P.30 コード 8 InvalidateRect(hWnd,NULL,TRUE);の前に追加）

```
for(i = 0 ; i < MAX_ENEMY ; i++)
{
    if(mEnemy[i] != NULL)
    {
        if(mEnemy[i]->Destroy == 1)
        {
            DestroyEnemy(mEnemy[i]);
            mEnemy[i] = NULL;
        }
    }
}
```

コード 19（P.27 コード 4 KillTimer(hWnd,TIMER_ID);の後に追加）

```
for(i = 0 ; i < MAX_ENEMY ; i++)
{
    if(mEnemy[i] != NULL)
    {
        DestroyEnemy(mEnemy[i]);
        mEnemy[i] = NULL;
    }
}
```

削除をするときは NULL でない要素を探します（コード 18）。また、機体が破壊されている場合、Destroy 変数の値が 1 になるので、削除を行います。よって、NULL でない要素でか

つ Destroy フラグが 1 であれば、削除を行います。これには、先ほど作成した DestroyEnemy 関数を使います。削除をしたら、削除した要素は NULL にしておきましょう。そうしないと、この要素を参照したときに中身がないのに参照することになり、エラーの原因となります。

ゲームが終了するときの削除も同じです（コード 19）。

3. 敵の表示

敵の作成が終わったので、仕上げに敵の表示を行います。今回も画面の表示はテキストで行います。前回とあまり変わらないので、最初にコードを見てみましょう。

コード 20（P.25 115 行目の後に追加）

```
for(i = 0 ; i < MAX_ENEMY ; i++)
{
        if(mEnemy[i] != NULL)
        {
                TextOut(hdc,mEnemy[i]->Position.x,mEnemy[i]->Position.y,"▼",2);
        }
}
```

今回は「▼」を使って敵を表示をしています。表示をするために描画する際も、敵のチェインの要素が NULL でないか、チェックします。

さて、これで一通り終わりました。F5 キーでコンパイルして実行してみましょう。点々と敵が表示されていく様子が見られれば完了です。コンパイルエラーが出てしまう場合、もう一度見直してみましょう。

今回は、2 回目ということで、5 章、6 章の説明をしました。本当は 7 章も説明をしようと思ったのですが、予想以上にページ数が増えてしまったので、ここで今回は終わります。

○前回の記事での間違いについて。

前回の記事で、Visual C++ Express Edition で Platform SDK を使う方法を掲載しましたが、掲載されているページ通りに設定しても「Win32 プロジェクト」が作成されないことがわかりました。申し訳ありません。

「Win32 プロジェクト」を表示させるようにするには、こちらの方を参照してください。

Visual C++ Express について　　http://lets-go.hp.infoseek.co.jp/c_setting09.html

今回の記事については、こちらでも公開します。また、記事中に出てきたソースコードについては、章ごとにダウンロードできるようになっています。

Web **ページ**: http://yasuharu.net/word/shooting/2/

間違いの指摘などありましたら、以下の連絡先まで遠慮無く連絡してください。

メールアドレス: word@yasuharu.net

ようこそ☆ WorDonald へ

文 編集部 mitty

やあ（´・ω・`）
ようこそ、WorDonald へ。

このポテトはサービスだから、まず食べて落ち着いて欲しい。

うん、「また」なんだ。済まない。
~~やすし様がみてるしね~~、謝って許してもらおうとも思っていない。

でも、このポテトを見たとき、君は、きっと言葉では言い表せない「ときめき」みたいなものを感じてくれたと思う。
殺伐とした世の中で、そういう気持ちを忘れないで欲しい
そう思って、このポテトを作ったんだ。

じゃあ、ポテトを揚げようか。

「WORD でポテトパーティをやるらしい」

現思特講でなし崩し的に(?)WORD 編集部員となった私がその言葉を耳にしたときに思ったのは、「WORD がただのポテトなんかに興味があるはずがない…！」ということでした。

とは言っても、高校現役時代は「その体のどこに入っているんだ」「mitty の『普通』って **3 人前**だよね」と、行く先々で驚き呆れられたきた手前、まぁ大丈夫だろうとの安易な気持ちで参加することにした、夏休みも始まったばかりの 7 月 3 日。

文字通り**胃がむかつくような惨劇**が待っているとも知らず、私はその他数名の一年生とともに 3C 棟ラウンジに向かったのでした。

未知との遭遇

バグ祭備品庫から鍋、WORD 部屋からコンロと調理器具を発掘し、ポテト買い出し部隊を待つこと数十分。「朝飯食ってないから早く食いたいよ」などとじりじりと待っていた編集部員たちに待望の一言が。

「ポテト買ってきたよー」

待ってました、とばかりに振り向いた全員の表情が固まる。

なんだこれは…

ポテト耐久 15kg

段ボールいっぱいの…って一箱に入りきらないので机に並べてみました。
…どう考えても多いよ！
ここにいるのは 7 人だから約 2kg/人…。

これは良い死亡フラグですね。

まぁやってしまったことは仕方ないので、諦めて食べる準備をします。

準備

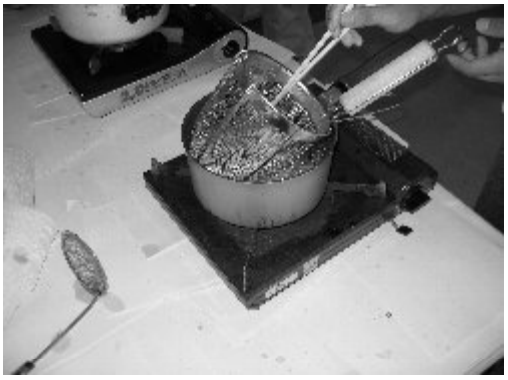
まず、机に鍋を並べて、油を入れる前に水気を飛ばす。

贅沢に 3 スレッド…というかこの場合は 3 コア(?)使って処理。
ヘテロコアだけど。

とか何とか下らないことを言っている間にも、油やら食塩やらケチャップやら着々と用意は進んでいく。爽健美茶が凍っているのはご愛敬。

お茶も箱買いとか…どんだけ(ry

いよいよ投下



さて、油も温まってきたので揚げることにします。なんか、**どう見てもラーメンとかを茹でる用途な柄付きのざる**ですが、使えればいいのです。

というか、もうこの際**揚げられれば方法なんてどうでもいいよね。**

油を追加。とにかく難しいことは考えずにひたすら揚げる簡単なお仕事です。

そうこうしているうちに綺麗な狐色に。白黒なのでお見せできませんが…。

そして、おそるおそる口をつけると…

「…普通に美味しいじゃん」

何もつけなくてもかなりいけます。本当、普通に美味しいじゃないですか、ポテト。最初は**「この量あり得ないって」「これでまずかったら大惨事だな…」**とか言っていたことなど忘れてしまったかのようにむさぼり食う編集部員。

まぁ、腹が減ってればどんなものでも美味しいですしね。

WorDonald 誕生

さて、皆の腹が段々と満たされてきた頃、とある編集部員が「そういえばポテトって揚げるとどのくらい軽くなるんだろ」と呟きました。

そんなの量ってみれば良い、ということで早速計量器が用意され 200g きっかりに量り取られたポテトを煮えたぎる鍋へと投下。

きっかり 7 分後。きれいに揚がったポテトの重さは 137.5g。結構軽くなるものですね…。

before

after

そして、「種類によって掛かる時間が違うのかな」「細い方が速そうだ」「そんなのいいから早く食え」「ポテトウマー」などとラウンジに穏やかな空気が漂い始めていたその時。

「これって MacD ● nald より安くて美味いんじゃね？」

ソ・レ・ダ！

と言うわけで、ポテトだけ S・M・L すべて買うとかいう実に馬鹿なことになったわけですが(きっと店員さんから奇異な目で見られたに違いない、絶対そうだ)、**15kg** というポテトを前にして頭がおかしくなった編集部員が**今更そんなことを気にするわけもなく。**

S Mi Le そろったお！(頼んでないけど)

L：150g（270 円）

MacD ● nald のポテトの重さは、L が 150g （270 円）、M が 113g （220 円）、S が 90g （170 円）でした。対して冷凍ポテトは 1kg で 200 円…**これは大勝利じゃね？**

正確には油代とかガス代とかも計上しなければいけませんが、どう考えても 3 ～ 4 倍安いです。

「これは WorDonald として売り出したら元とれるんじゃないのか」とかいう冗談が現実味を帯びたり帯びなかったり。

M：113g（220 円）

S： 90g（170 円）

MacD ● nald のポテトは揚げが足りないので冷めると柔らかくなって美味しさが半減しますが、WORD ポテトはしっかり揚げているのでそんな問題ともおさらばです。**素晴らしい…。**

で、調子にのっていっぱいにした鍋を頑張って食べる編集部員たち。

うおお、まどろっこしいぜっ

……

　　_,,･･･,,,,_
／ ,’ 3　　`ヽｰっ
l　　　つ　　⌒_つ
　`’ｰ－－－-_,,,,,〃

無茶しやがって…

祭りの後

結局、15kg は食べきれませんでした。第一陣 7 人の他に、後から 3 人ほど加わって処理した結果、どうにか残り 1kg まで減らせたそうです。というかその 1kg はまだ WORD 部屋の冷凍庫に。

これが **WORD** クオリティ

書籍紹介

文 編集部 アスロンズ

第二回

書籍紹介とは

さて、このコーナーも第二回になりました。季節もいつの間にか秋になってきましたね。秋といえば読書の秋。書籍を読むには良い季節です。このコーナーでは、技術的に役立つものから知っておいて得をするものまで、コンピュータ関連の書籍をピックアップして紹介していきます。

月刊 NetworkWorld

第二回目は、月刊誌を取り上げます。今回取り上げた雑誌は「月刊 NetworkWorld」です。雑誌ということもあり、最新号である同 2007 年 10 月号を元に紹介させていただきます。

主力記事はネットワークソリューションの構築・運用・保守

同誌は全世界 21 カ国にて発行されています。内容に関しては各国様々となっています。その主力記事は、ネットワークに関する「製品・インフラ・最新技術・市場動向」など幅広い内容を取り扱っています。今回の 10 月号では、我々筑波大学も参加している「SiNET3」に関する記事もあり、非常に内容の濃い仕上がりになっています。主な読者対象は、ネットワークエンジニアやそれを目指す社会人・学生となっています。ある程度のネットワークに関する知識があれば、すんなりと理解することができると思います。

また、実際に販売されている製品と連動した広告企画もあり、現在市場で求められている内容に関しても触れることができます。

長期連載中の「追跡！ネットワークセキュリティ 24 時」というコーナーでは、実際に起こったインシデントなどをたとえ話として紹介されており、既に単行本も販売されています。

ネットワークが好きな人にお勧め

以上のような内容から、専門誌的な意味合いが高く広く一般受けはしないと思います。ですが、ネットワークが好きでしょうがない人や、SI やネットワークエンジニアとして活躍したい人にとっては、非常に納得のいく内容であると思います。

書籍名　：月間 NetworkWorld
著者名　：N/A
発　行　：株式会社 IDG ジャパン
WEB　：http://www.networkworld.jp/
ISBN　：N/A
値段　：950 円
頁数　：N/A
発行日　：毎月 18 日販売

同誌 2007 年 10 月号　公式 WEB サイトより引用

情報科学類誌

WORD

消えたA棟の謎号

発行者	情報科学類長
編集長	柳田 一樹
制作・編集	筑波大学情報学群 情報科学類誌WORD編集部 （第三エリアC棟212室）
印刷	第三学群棟印刷室
2007年9月22日	初版第1刷発行（386 部）
2007年10月19日	第2版第1刷発行（128 部）