第１章
Visual Basic プログラミングの基本

1.1　Riki-0 の実行ファイルの画面を見てみよう

●実行ファイル Riki-0.exe を立ち上げる

①パソコンの電源スイッチを入れ，Windows を起動してデスクトップ画面を出し，付属の CD-ROM を CD ドライブにセットする（以降の説明は，Windows98 の環境で，VisualBasic6.0 がインストールされた状態を前提としている）．

②デスクトップ上の「マイコンピュータ」**アイコン**をダブルクリックして「マイコンピュータ」のウィンドウを開き，その中の CD-ROM ドライブをダブルクリックして，CD-ROM ドライブのウィンドウを開く．

③このウィンドウには，収録されている二つの**フォルダ**の `Vb6.0-Source` と `Vb6.0-exe` のアイコンがあるので，**実行ファイル**（exe file）のフォルダである後者をダブルクリックして，`Vb6.0-exe` のウィンドウを開く．

④フォルダ `Vb6.0-exe` のウィンドウには，5 個の下位フォルダ `Riki.exe`，`Nami.exe`，`Denji.exe`，`Atom.exe`，`DataDeal.exe` があるので，`Riki.exe` のアイコンを選んでダブルクリックする．

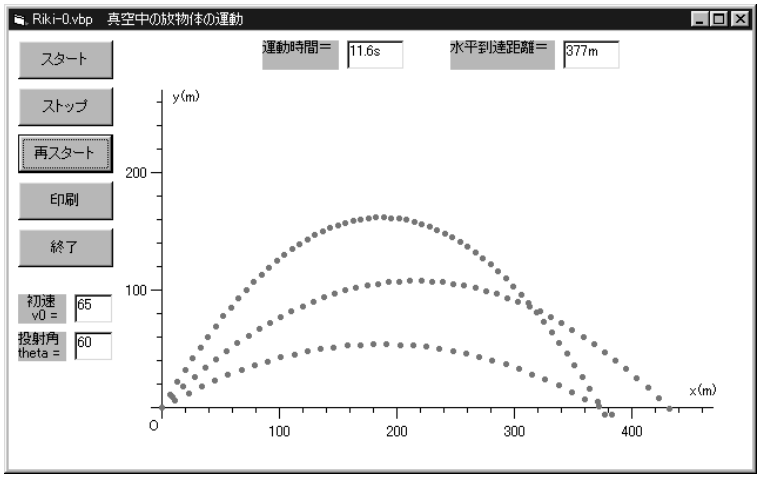
⑤`Riki.exe` のウィンドウには運動力学の実行ファイルのアイコンが 15 個並んでいる．この中の `Riki-0.exe` が，これから見ようとする「真空中の放物体の運動」の実行ファイルである．

以上の操作の順をそれぞれのアイコンで示すと下の図になる．

●実行ファイル Riki-0.exe を実行してみよう

①`Riki-0.exe` のアイコンをダブルクリックすると実行画面が現れるが，このままでは画面は何も変化しない．ユーザーの働きかけを待機している状態である．

〈図 1.1〉真空中の放物体の運動（Riki-0.exe）



②まずスタートボタンにマウスカーソルを合わせてクリックすると，下のボックスに示された初速v0=65m/s，投射角theta=30°のときの放物体の運動の軌跡が描かれる**（図1.1）**．この両値はユーザーが与えなくてもすでに定まっているので，**デフォルト**（default）値（**既定値**）と呼ばれる．
③そこで投射角thetaを45°あるいは60°に変えて再スタートボタンをクリックすると，それぞれの場合の放物体の軌跡が画面に追加される．また運動の途中でストップボタンをクリックすると描画が停止し，スタートボタンをクリックすると描画が継続される．また初速v0の値も変えて再スタートさせてみる．図1.1の上部にある運動時間や水平到達距離の値はそれぞれの初速，投射角の場合に対応した値を示す．
④また描画を終えた静止画の状態で印刷ボタンをクリックすると，（プリンタが接続されていれば）印刷が始まる．終了ボタンをクリックすると`Riki.exe`のウィンドウに戻る．

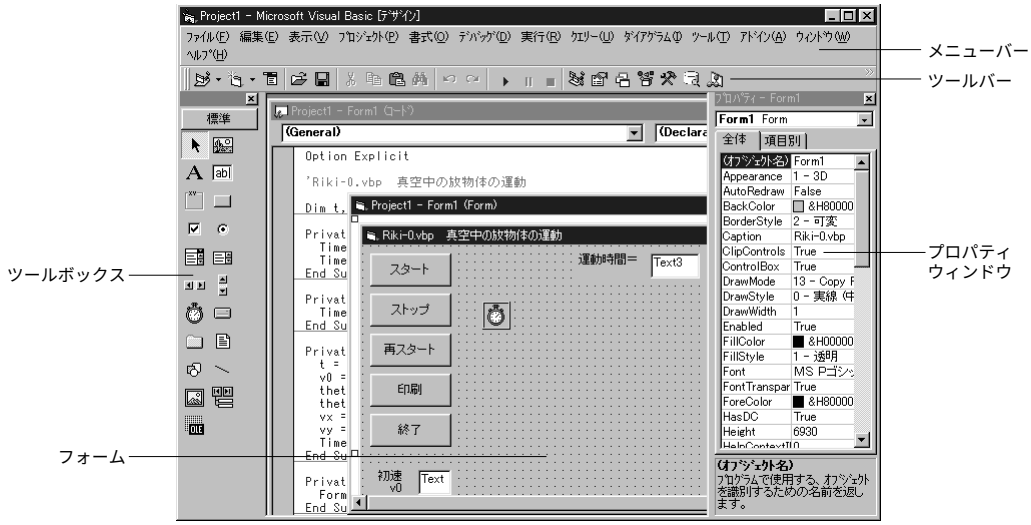
＊　　＊

このようにVisual Basicでは，ユーザーが「スタート」などのコマンドボタンをクリックするという働きかけに応じてパソコンが反応するしくみになっているので，これを**イベントドリブン**（event driven）と呼んでいる．スタートボタンのクリックという出来事（event）に対して，運動体を一定時間ごとに描画するためのタイマ機能が作動する（driven）ということである．

1.2　Visual Basicによるプログラミングの方法

前節の実行ファイルでは，インプット（コマンドボタンのクリック，初速や投射角の入力）に対してアウトプット（放物体の軌跡のグラフ）が得られただけで，コマンドをパソコンに伝えて実行させるためのプログラムリストについてはまったく見ることができなかった．この節では，コマンドボタンを画面に配置し機能させるためのオブジェクト画面の準備やVisual Basicプログラムの作成法について説明する．

〈図 1.2〉オブジェクト画面とプログラムリスト



●まず Riki-0.vbp を開いてみよう

①Riki-0.exeを立ち上げたときのように，「マイコンピュータ」→CD-ROMドライブ→Vb6.0-Source→Riki→Riki-0.vbpの順にアイコンをダブルクリックすると，Riki-0.vbpのプログラムリストが表示される．

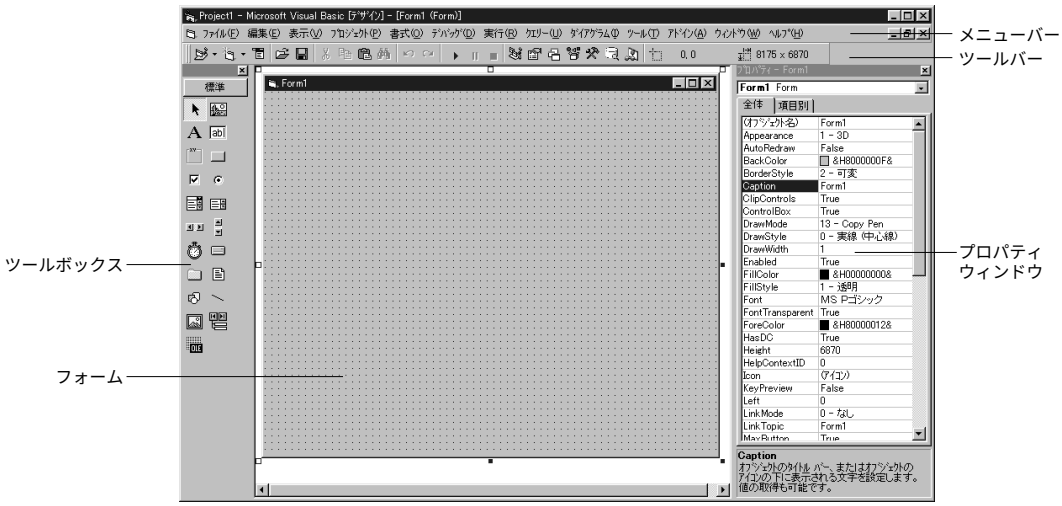
②続けてメニューバーの実行（R），開始（S）の順にクリックすると，Riki-0.exeのときと同じ実行画面が得られ，**図1.1**と同様の放物運動のシミュレーションを実行することができる．

③この画面の左上の角にマウスカーソルを合わせてマウスボタンを押したまま右下にずらせて（ドラッグして）指を離す（ドロップする）．移動させた背後にプログラムリストが見られる．

④次に，画面の「終了」ボタンをクリックしてプログラムリストに戻し，メニューバーの表示（V），オブジェクト（B）の順にクリックするとプログラムリストを覆って**オブジェクト画面**が表れる（**図1.2**）．この画面は時計マークが入っている点を除いて，**図1.1**の実行画面とよく対応している．その左側は**ツールボックス**（Tool Box）で，**フォーム**（Form）上に**コントロール**（コマンドボタンやテキストボックスなど）を貼り付けるために設けられている．右側は**プロパティウィンドウ**（Property window）で**図1.2**のようにプログラム名を記入したり，貼り付けたコマンドやテキストラベルに機能名や属性名などを記入するための一覧表である．具体的にはあとで説明する．

以上に見たように，Visual Basicでは，数値の代入はBASICと同様キー入力で行うが，それ以外の操作はマウスで画面上のコマンドボタンをクリックすることで実行するので，コマンドボタンをフォーム上に貼り付ける仕事（**フォームデザイン**）が余分に加わる．BASICでは，スタート（RUN）や一時停止などのほか，画面上の指示にしたがう入力や選択などすべてキーボード上で行い，マウスの利用は特別な場合以外は考えられなかった．その他の違いについては後節で述べることにする．

〈図 1.3〉統合開発環境



●オブジェクト画面（ユーザーインターフェース）の作成

あらためて，Windowsの立ち上げからVisual Basic 6.0を開く場合の手順を説明する．デスクトップの「スタート」をクリックした後，プログラム（P）→Microsoft Visual Basic 6.0→Microsoft Visual Basic 6.0の順に選択してクリックすると「新しいプロジェクト」と表題のついたウィンドウが出る．標準EXEが選択されているのを確認して「開く」をクリックすると**図1.3**の**統合開発環境**が現れる．ここからオブジェクト画面の作成に取りかかることになる．

▼オブジェクト画面の作成の手順

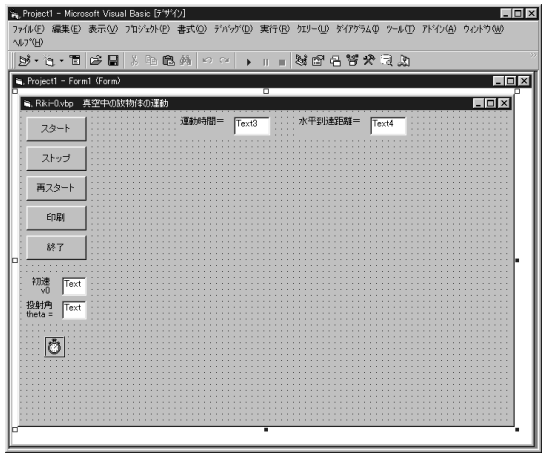
①まず，コマンドボタンをフォーム上に5個貼り付ける．ツールボックスのコマンドボタンをマウスでダブルクリックするとフォーム上に移り，これをさらに配置したい場所までドラッグしてドロップする．同様に残る4個のコマンドボタンを貼り付ける．

②次に「初速v0 =」と「投射角 theta =」を記入するラベル（A）と数値を記入あるいは代入するテキストボックス（ab）をフォームに貼り付ける．同様に，「運動時間 =」と「水平到達距離 =」を記入するラベル（A）とその計算結果を出力するテキストボックス（ab）をフォームに貼り付ける．これらのボックスの大きさは適当に伸縮できる．

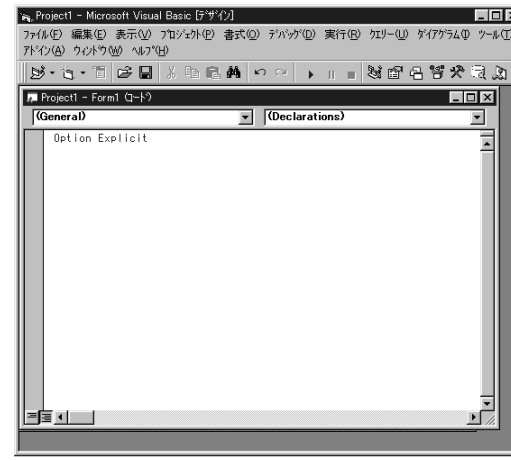
③最後に時計マークをフォームに貼り付ける．

以上でオブジェクト画面はできあがりである．このように，ユーザーが必要と考える**コントロール**（コマンドボタン，代入用テキストボックス，出力用テキストボックスなどの総称）を選択しフォーム上に配置することにより，ユーザーはそれらを対象に操作（クリックや入力）し，それに対応したプログラムを通してパソコンに働きかけるのである．このように，オブジェクト画面はユーザーとパソコンを仲介するので，**ユーザーインターフェース**と呼ばれる（参考文献2），p.199）．

〈図 1.4〉プロパティの設定を終えたオブジェクト画面



〈図 1.5〉コードエディタウィンドウ



●プロパティの設定

①ツールバーの「プロパティウィンドウ」ボタンをクリックしてウィンドウを開き，**キャプション**（caption）欄のForm1を消し，ここにプロジェクト名「Riki-0.vbp　真空中の放物体の運動」と書き入れると，フォームの最上段のキャプションバーに表示される．

②次は，フォーム上に配置したコマンドにそれぞれの役割を示す**プロパティ名**（スタート，ストップ，再スタート，印刷，終了）を上のプロジェクト名のときと同様にプロパティウィンドウのキャプション欄への記入を通して書き込む．ラベルについても同様に書き込む．プロパティの設定を終えたオブジェクト画面を**図1.4**に示す．

●プログラムコードの記述

オブジェクト画面に必要なコントロールを配置し，それに加えたクリック操作をパソコンに伝えて意図した結果を引き出すのがプログラムの役割である．そのためプログラムは，スタート，ストップ，再スタート，印刷，終了などのコマンドに対応した各ユニットプログラム，シミュレーション画像の場合に必要な条件（描画の単位や時間単位の設定，初期条件，色指定など）を含むユニットプログラム，時間的な変化を計算しグラフ表示するユニットプログラムなどの組み合わせで構成される．これらの独立したユニットプログラムは**プロシージャ**(procedure，手続きの意)と呼ばれる．

まず，メニューバーの「表示（V)」,「コード（C)」の順にクリックして，オブジェクト画面から**コードエディタウィンドウ**（code editor window）**（図1.5）**に切り替える．

プログラムコードの打ち込み方は，BASICのときと同様にキー入力するが，Form1.やTimer1.と打てばそのあとの候補文字が選べる「自動メンバー表示」と「プロシージャを連続的にコーディングする機能」を事前に設定しておく（参考文献1)：p.17，参考文献2)：p.32)．

①自動メンバー表示を設定する

②同じコードエディタウィンドウにすべてのプロシージャを表示させる

「ツール（T)」をクリックして「ツールメニュー」を開き，その中の「オプション（O)」をクリックして「オプション」ダイアログボックス（dialogue box）を表示させる（**図1.6**).

「編集」をクリックして「編集」に関する選択表を選び，その中の「自動メンバー表示（L)」と「モジュール全体を連続表示（M)」のチェックボックスをオンにする.

①の「自動メンバー表示」の設定はめんどうなスペリングをせずに済むのと，スペルミスを防ぐのに役立つ．Form1.を付けたため行が長すぎて困るという場合は，Form1.を消して行を縮めるか，最初からこの機能を外して手打ちをすればよい．しかしForm1だけでなくForm2も同時に使うときは，それぞれForm1.，Form2.を付ける必要がある（その例はDenji-6.vbpやDenji-7.vbpに見られる).

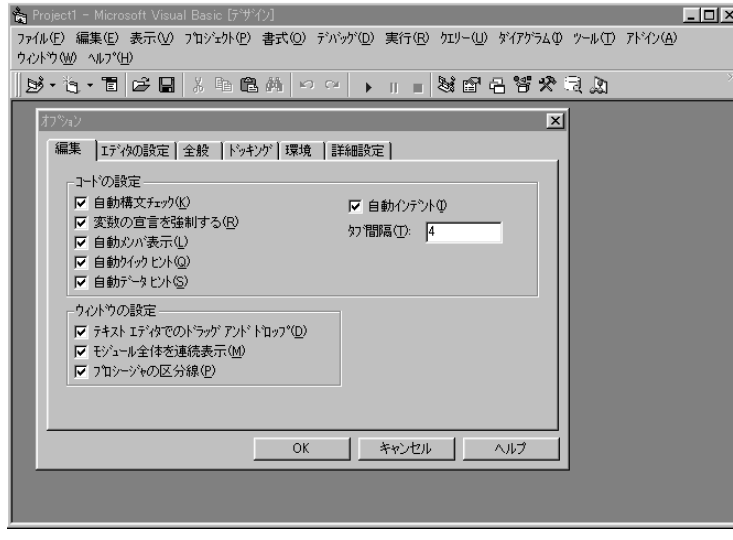
②の機能を具体的にいえば，**図1.5**のエディタウィンドウのいちばん上の"General"が入っているワクの横の▼をクリックすると，先にオブジェクト画面に貼り付けたCommand1～5やForm，Timer1などのプロシージャを作る必要のある項目が表れる．たとえば最初のCommand1をクリックすると，エディタウィンドウに次のようなCommand1のプロシージャの最初と最後の行が表示される：

```
Private Sub Command1_Click()
    ...
End Sub
```

この空いている行に必要な行文を埋める．同様に他のコマンドやForm_Load，Timer1のプロシージャを作成していけば，同じコードウィンドウに全プロシージャを連続表示したプログラムができあがる.

このようにして作成した**プログラムリスト**（Riki-0.vbp）を**リスト1.1**に示す.

〈図 1.6〉「オプション」ダイアログボックス



〈リスト 1.1〉真空中の放物体の運動（Riki-0.vbp）

```
Dim t, dt, vx, vy, x, y, xF, yF, v0, theta, g, Pi, red, white, i

Private Sub Command1_Click()                                    'スタート
  Timer1.Interval = 1                                           'Timer プロシージャを実行させる時間間隔を1ms
  Timer1.Enabled = True                                         'Timer のスタート命令
End Sub

Private Sub Command2_Click()                                    'ストップ
  Timer1.Enabled = False
End Sub

Private Sub Command3_Click()                                    '再スタート
  t = 0: x = 0: y = 0                                           '初期条件
  v0 = Val(Form1.Text1.Text)                                    '初速の入力
  theta = Val(Form1.Text2.Text)                                 '投射角(度)の入力
  theta = theta * Pi / 180                                      '投射角を rad 単位に直す
  vx = v0 * Cos(theta)                                          '初速度の水平(x)成分
  vy = v0 * Sin(theta)                                          '初速度の鉛直(y)成分
  Timer1.Enabled = True                                         'タイマをスタート
End Sub

Private Sub Command4_Click()                                    '印刷
  Form1.PrintForm
End Sub

Private Sub Command5_Click()                                    '終了
  End
End Sub

Private Sub Form_Load()
  Form1.Cls                                                     '画面クリア
  Form1.ScaleMode = 3                                           '長さの単位を１ピクセル
  Form1.AutoRedraw = True                                       '自動再表示命令「詳細：本文)
  dt = 0.2: g = 9.8: Pi = 3.14159                               'dt=時間区分、g=重力加速度
  v0 = 65: Form1.Text1.Text = v0                                '初速 v0(m/s)の既定値
  theta = 30: Form1.Text2.Text = theta                          '投射角(度)の既定値
  theta = theta * Pi / 180                                      '投射角を rad 単位に直す
  vx = v0 * Cos(theta)                                          '初速度の水平(x)成分
  vy = v0 * Sin(theta)                                          '初速度の鉛直(y)成分
  red = QBColor(12)                                             '赤色
  white = QBColor(15): Form1.BackColor = white                  'バックの色を白
  Zahyo
  Form1.FillStyle = 0: Form1.FillColor = red                    '赤色で塗りつぶし命令
  Form1.Circle (130, 320), 2, red                               '初期の放物体の表示
End Sub

Private Sub Timer1_Timer()
  t = t + dt
  x = vx * t: y = vy * t - 1 / 2 * g * t ^ 2                    '放物体の位置の x,y 座標値
  xF = 130 + x: yF = 320 - y                                    '運動体の位置のフォーム座標
  Form1.Circle (xF, yF), 2, red                                 '放物体の表示
  If y <= 0 Then                                                '着地したら
    Timer1.Enabled = False                                      'タイマを止めて
    Form1.Text3.Text = 0.1 * Int(10 * t + 0.5) & "s"            '着地までの運動時間の表示
    Form1.Text4.Text = Int(x + 0.5) & "m"                       '水平到達距離の表示
  End If
End Sub

Private Sub Zahyo()                                             '座標軸と目盛り
```

〈リスト 1.1〉真空中の放物体の運動（Riki-0.vbp）（つづき）

```
  Form1.Line (130, 320)-(600, 320): Line (130, 50)-(130, 320)     'x,y軸
  CurrentX = 120: CurrentY = 330: Print "O"
  CurrentX = 580: CurrentY = 300: Print "x(m)"
  CurrentX = 140: CurrentY = 50: Print "y(m)"
  For i = 0 To 23                                                 'x軸の20m目盛り
    Form1.Line (130 + 20 * i, 320)-(130 + 20 * i, 325): Next i
  For i = 0 To 4                                                  'x軸の100m目盛り
    Form1.Line (130 + 100 * i, 320)-(130 + 100 * i, 330)
    CurrentX = 122 + 100 * i: CurrentY = 335: Print Format(100 * i, "###"): Next i
  For i = 0 To 13                                                 'y軸の20m目盛り
    Form1.Line (130, 320 - 20 * i)-(125, 320 - 20 * i): Next i
  For i = 0 To 2                                                  'y軸の100m目盛り
    Form1.Line (130, 320 - 100 * i)-(120, 320 - 100 * i)
    CurrentX = 100: CurrentY = 315 - 100 * i: Print Format(100 * i, "###"): Next i
End Sub
```

1.3　プログラム（Riki-0.vbp）の解説

●変数などの宣言

まず1行目で，Dimによりプログラムの中で使われるすべての変数，定数，色を宣言し，メモリ領域を確保しておく．これをしていない変数を使うとエラーとなる（BASICでは不要だった）．

●コマンド（Command1 ～ 5）プロシージャ

Command1の「スタート」命令のプロシージャの中に二つの命令が入っている．

Timer1.Interval = 1は，下にあるTimer1プロシージャを1msの時間間隔で実行させる命令である

Timer1.Enabled = Trueは，Timer1をスタートさせる命令である

これらの命令はオブジェクト画面を作成したときにツールボックスから時計マークを貼り付けたことによって機能している．

次の「ストップ」命令は上のスタート命令と逆なので，TrueをFalseに変えればよい．

```
Timer1.Enabled = False
```

次の「再スタート」命令では，最初のスタート時と初速v0，投射角thetaのいずれかまたは両方の値を変えて投射するので，初期条件（時刻t＝0に原点x＝0，y＝0から打ち出す）の設定，初速v0値や投射角theta値の代入，それによる初速度のx，y成分の計算を行った後，Timer1をスタートさせる．

「印刷」，「終了」プロシージャの中のプリント命令Form1.PrintFormやEndは，それぞれのコマンドボタンをクリックすることにより機能する．

● Form_Load プロシージャ

Form_Loadプロシージャでは，Timer1プロシージャを実行して運動体の軌跡を描くのに必要な条件設定や諸準備（長さの単位，時間区分，初速や投射角の既定値の設定，色の指定，Zahyoプロシージャの呼び出し，初期の運動体の表示）を行っておく．以下に具体的に述べる．

Form1.ClsはBASICにもあったクリア命令CLS　3に相当し，Form1上のグラフィックスとテキストの両方を消す命令である．

Form1.ScaleMode = 3 は，Form1上の長さの単位を**ピクセル**(最小単位でドットに同じ)にする．

Form1.AutoRedraw = True は，Form全体の画像がメモリに保持され，必要なときにいつでも再表示されるためのもので，以降すべてのプログラムに入れている．

dt=0.2sは放物体を表示する位置を計算する時間区分，g = 9.8 m/s^2 は重力加速度，Piは円周率πである．

次の2行は，v0 = 65m/s とtheta=30°を初速と投射角の既定値（デフォルト値）として与える命令である．

theta=theta*Pi/180は，パソコンプログラムでは角度をラジアン(rad）単位で扱うので，theta(度)*Pi/180によりrad単位に換算して，theta(rad）として使うための換算式である．

このrad単位に換算した投射角thetaを用い，次の2式で初速度のx，y成分を求める．

```
vx = v0*Cos(theta): vy = v0*Sin(theta)
```

red = QBColor(12)は赤色を定義し放物体の色に用いる．またwhite = QBColor(15)は白色を定義し，Form1.BackColor = whiteにより画面の背景色に用いる．

次の行のZahyoは，放物体の運動の軌跡を描画するのに必要な目盛り付き座標軸が下にZahyoプロシージャとして定義してあるので，ここに引き出して表示するためのものである．

Form1.FillStyle＝0は閉じた枠内を塗りつぶす命令で，Form1.FillColor＝redはその塗りつぶし色を赤に指定する命令である．

Form1.Circle(130,320)，2，redは初期（t＝0）の放物体をフォーム座標点（130，320）を中心に半径2ピクセルで赤色縁の円で表示する命令で，その前の命令と合わせて塗りつぶされた赤丸になる．

● Timer1 プロシージャ

これは，運動体の刻々の動きを描画するためのプロシージャである．

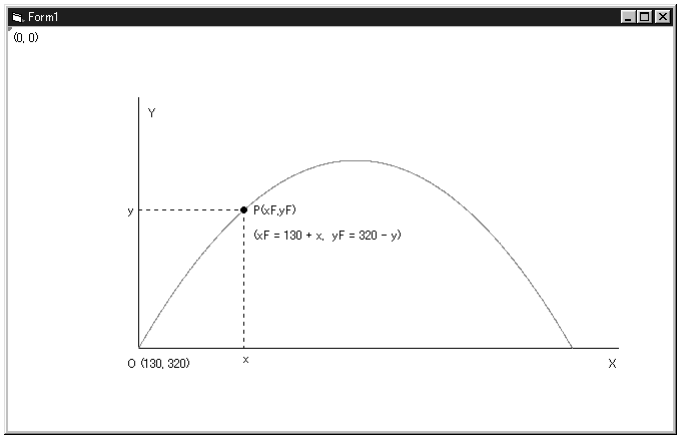
時刻tの運動体の位置（x, y）は，$x = v_x t$ ，$y = v_y t - 1/2gt^2$ で表されるから，時間dtごとの位置に運動体を半径2ピクセルの赤丸で表示する命令は，

```
x = vx*t : y = vy*t  -  1/2*g*t^2
xF = 130 + x : yF = 320  -  y
Form1.Circle (xF, yF), 2, red
```

となる．これをCommand1のスタートプロシージャにより時間間隔1msで繰り返し実行を指令される．

BASICの場合のオリジナルスクリーン座標と同様，**フォーム座標**は左上隅を原点とする．これと適当な位置（このプログラムではフォーム座標上の点（130, 320））に原点を置く**物理座標**の関係は，**図1.7**のようになる．Circle命令やPSet命令で指定する座標点はフォーム座標によるので，物理座標点（x, y）をフォーム座標（xF, yF）（このプログラム例では（xF = 130 + x，yF = 320 – y））に換算して用いなければならない．ここでxFやyFのFはFormの頭文字を意味している．

〈図 1.7〉フォーム座標と物理座標



y<=0，すなわち着地したら**Timer1**を停止させ，そのときまでの時間を小数第2位で四捨五入し，水平到達距離を小数第1位で四捨五入してそれぞれのテキストボックスに表示する．ここで，If... Then型条件分岐を利用している:

```
If y <= 0 Then                                    '着地したら
Timer1.Enabled = False                            'タイマを止めて
  Form1.Text3.Text = 0.1*Int (10*t +0.5) & "s"
                                                  't値を小数第2位で四捨五入して表示
  Form1.Text4.Text = Int (x + 0.5) & "m"          'x値を小数第1位で四捨五入して表示
End If
```

●座標（Zahyo）プロシージャ

ここではフォーム座標点（130, 320）に原点をおく物理座標のx，y軸を設定し，For 〜 Nextを使って両軸の20m，100m目盛りと100m目盛り数値をつける．CurrentX，CurrentYは文字をプリントするときの位置を指示するもので，BASICのLOCATEに相当するが，文字位置の指定が図形と共通のスケールで行える点が便利である．

このようにユーザーが適当な名前（Zahyo）でプロシージャを作成しておけば，その名前で必要な場所で使うことができる．プログラムの最初に"Option　Explicit"が自動的に表示される．これはすべての変数の宣言を強制するステートメント（statement）でプログラムの動作に影響しないが，宣言忘れの変数を指摘してくれるので初心者にはありがたい．これを除去したいときは，メニューバーの「ツール」→「オプション」をクリックしてオプションウィンドウを開き，その中の「変数の宣言を強制する（R）」のチェックをオフにする．"Option Explicit"がオンのときは，For 〜 Next の中の変数 iの宣言も忘れないよう気をつける．

1.4　プログラムの保存と実行

一つや二つのプログラムであれば既存の適当なフォルダのファイルとして保存してもよいが，本書のように系統的にプログラムを作成するのであれば，前もってツリー（tree）構造にフォルダを作成し，そこにプログラムファイルとして整理・保存することをおすすめしたい．本書に付属のCD-ROMの中にそのようなフォルダ`Vb6.0-Source`，`Vb6.0-exe`が設けられていることは1.1節で説明したが，以下，パソコンにフォルダを作成するところから説明する（CD付属のコピーメニューを使うこともできるが，コピーメニューを使わないやり方も紹介しておく）．

①フォルダの作成

デスクトップの「マイコンピュータ」→Windows98(C)の順にクリックして，Windows98(C)のウインドウを開く．空いた場所を右クリックするとメニューが開くので，「新規作成(N)」→「フォルダ(F)」を選んでクリックすると，「新しいフォルダ」と書かれたフォルダができる．文字「新しいフォルダ」を消して読者が希望する名（ここでは例として「Vb6.0-Source」としておく）に書き換える．これで新しいフォルダVb6.0-Sourceができあがったので，これを適当な場所にドラッグしておく．

次にフォルダVb6.0-Sourceをクリックすると，そのウィンドウが開く．ここに次々に右クリックして，上と同じ要領で新しい名前`Riki`，`Nami`，`Denji`，`Atom`，`DataDeal`をつけたフォルダを作成する．

②Rikiフォルダに Riki-0.vbp ファイルを保存する

Riki-0.vbpのプログラムを作成した後，メニューバーの「ファイル（F）」→「名前を付けてプロジェクトの保存（E）」の順にクリックして，「名前を付けてファイルの保存」のウィンドウを開く．「保存する場所（I）」の▼をクリックしてWindous98（C）→Vb6.0-Source→Rikiの順にクリックして保存する場所のフォルダを`Riki`にする．下のほうにあるファイル名（N）の欄にフォーム名`Form1.frm`に替えて`Riki-0.frm`と記入し，「保存（S）」ボタンをクリックする．

続いて「名前を付けてプロジェクトの保存」のウィンドウが表示されるので，このウィンドウのファイル名（N）の欄にプロジェクト名`Project1.vbp`に替えて`Riki-0.vbp`と記入し，「保存（S）」ボタンをクリックする．これでこのプログラムの保存が完了する．

以上の`Riki-0.vbp`の保存の流れから，`Riki-0.vbp`のツリー構造的な位置づけは次のように表される：　`C¥Vb6.0-Source¥Riki¥Riki-0.vbp`

③プログラムの保存の確認・読み込み・実行

メニューバーの「ファイル（F）」→「プロジェクトを開く（O）」の順にクリックして「プロジェクトを開く」のウィンドウを開く．ファイルの場所（I）の▼をクリックして，順にWindows98（C）→Vb6.0-Source→Rikiとクリックすると，Rikiフォルダのウィンドウが開かれ，その中にファイル`Riki-0.vbp`を確認することができる．このファイルRiki-0.vbpをクリックすると下のファイル名（N）欄に`Riki-0.vbp`と出るので，「開く（O）」をクリックすると，Riki-0.vbpのプログラムリストが表れる．メニューバーの「実行（R）」→「開始（S）」をクリックすると実行画面が現れ，「スタート」ボタンをクリックして放物運動のシミュレーションを始めることができる．

1.5　本書で多用する命令

以降に展開する他のプログラムもRiki-0.vbpと似た構造で，使われる命令用語に共通するものが多い．読者がプログラムの作成にあたって参照しやすいように，まとめておく．

①Timer1実行の時間間隔の設定

（例）1msの場合：Timer1.Interval = 1，100msの場合： Timer1.Interval = 100

②長さの最小単位の設定

（例）1ピクセル（ドット）の場合：Form1.ScaleMode = 3

③自動再表示命令：　Form1.AutoRedraw　=True

プロパティAutoRedrawがTrueに設定されている場合のみ，グラフィックスは自動的に再描画される．したがって本書のすべてのシミュレーションプログラムにはこの命令が必要である．

④色の指定方法1

明るい赤：red = QBColor(12)，明るい青：blue = QBColor(9)，明るい緑：green = QBColor(10)，黒：black = QBColor(0)，白：white = QBColor(15)，緑：green = QBColor(2)

⑤色の指定法2（この場合，画面の色数を多色のTrue Color（24または32ビット）に設定しておく）

同じ色を濃淡を変えて使う場合（例：Nami-16.vbp，Atom-6.vbp）に効果的である．

（例）もっとも明るい赤色：red = RGB(255,0,0)，　もっとも明るい黄色： Yellow = RGB(255,255,0)

R（赤），G（緑），B（青）を3原色とする混合法で色を合成し，カッコ内の数（0～255）が大きいほどその色が明るく，小さいほど明度が低くなる．

⑥画面の背景色を白色にする．

```
white = QBColor(15):Form1.BackColor = white
```

（例外）Nami-16.vbp（光の回折像）の場合は背景色をプロパティウィンドウにあるカラーパレットを使って薄い灰色に選んである．

⑦円内や長方形内の色の塗りつぶし命令と解除

塗りつぶし：　Form1.FillStyle = 0 ，　塗りつぶし解除：　Form1.FillStyle = 1

⑧フォーム座標内の位置（xF，yF）に赤色の点を打つ

```
Form1.PSet(xF,yF),red
```

⑨2点（x1F，y1F），（x2F，y2F）の間を青色の線分で結ぶ

```
Form1.Line(x1F,y1F)-(x2F,y2F),blue
```

⑩点線の指示

（例）2点A（$x1$, $y1$），B（$x2$, $y2$）を赤い点線で結ぶ場合

点線を指示：　　Form1.DrawStyle = 2

赤線で結ぶ：　　Form1.Line(x1,y1)－(x2,y2),red

実線に戻す指示：　Form1.DrawStyle = 0

⑪フォームの横幅の表示：Form1.ScaleWidth，フォームの高さの表示：Form1.ScaleHight

⑫ x の平方根を求める：　Sqr(x)

⑬文字を書く位置の指定法

フォーム座標で位置（140, 330）に物理座標の原点名 "O" を書く命令は，次のようになる．

```
Form1.CurrentX = 140: Form1.CurrentY = 330: Print "O"
```

⑭ Format 関数を用いて座標軸の目盛り数値を書く

（例）x 軸に 100m ごとに目盛りとその数値をつける命令は次のようになる．

```
For i= 0 To 4
  Form1.Line(130+100*i,320)－(130+100*i, 330)
  Form1.CurrentX=122+100*i:Form1.CurrentY=335: Print Format(100*i, "###")
Next i
```

（参考文献）山賀弘著，『親切でムリなく学ぶ Visual Basic4.0』，技術評論社，p.256

⑮四捨五入の方法

（例）x = 5.34 を小数第 2 位で四捨五入すると 5.3 になるが，VB や BASIC でこれを求めるには，小数点を切り捨てて整数化する命令 Int を用いて次のように計算する：0.1*Int(10*x + 0.5)=0.1*Int(53.4+0.5)=5.3

⑯v0=Val（Form1.Text1.Text）

本節のプログラム `Riki-0.vbp` の再スタートで，たとえばテキストボックス 1 に初速値 70m/s のつもりでキー入力した数字の 70 は文字列として受け取られるので，これを数値の 70 に変換して v0 値とする命令である．しかし実際の場合，Val をつけなくても済む場合も多いようである（`Riki-5.vbp` では必要だった）．本書では一応約束どおり，すべての場合に Val( )を使っておいた．

1.6　Visual Basic と BASIC の比較

①取り扱い上の比較

BASIC は初心者用といわれるようにわかりやすいが，機種によってプログラム言語が若干（とくにグラフィックスで）異なり，互換性に問題があった．32 ビット機と Windows の組み合わせ上で走る Visual Basic の出現でこの問題が解決されただけでなく，マウスによる画面操作化が進んで使いやすくなった．BASIC ではデータを INPUT して RUN キーを押してプログラムを実行したり，STOP キーなどを押して終了したりで，キーが操作の対象だった．Visual Basic では，画面に配置されたビジュアルな命令ボタンなどを対象（オブジェクト，object）にマウスのクリックで働きかける．したがって，命令ボタンをクリックしたら，それをどうコンピュータに伝え，どう応えさせるかの仲介役を果たすプログラムも当然 BASIC とは違ってくる．Visual Basic では，一つ一つの独立したオブジェクト操作に対応したユニット化されたプログラム（プロシージャ）を作成し，その組み合わせでプログラムを構成するので見やすい．これに対し BASIC では，一つのプログラムの中にすべてが入り込んで流れを作るので，段階的な仕切りとして注釈文を入れたり，`GOSUB`〜`RERURN` 命令を使ったりして見やすくする工夫をしても，構造化は難しい．

②プログラムの実行速度上の比較

BASICは1行ずつマシン語に変換して実行されるインタプリタ型言語なので，低スピードの16ビット機では問題があった．たとえば，波の進行のアニメーションを一定時間ごとの「書き消し法」で行おうとしても，一つ一つの波を瞬間的に描くことができないので，この方法はまず無理であった．そこでBASICのころ筆者がとった方法は，12ページに1/12周期ごとの波形を1個ずつ記録し，記録後一定時間をおいて1ページずつ表示する方法だった．現在ではこの原理は，ハードディスクに記憶し，1コマずつ再生する方法に生かされている．「書き消し法」は高速化した32ビット機のMS-DOS BASICで実行することもできるが，現在これを利用するユーザーはほとんどいないのではないだろうか．

Visual Basicプログラムを「読み込んだ」状態は，全体がマシン語に変換されてユーザーのはたらきかけを待つ状態にあるので，「スタート」ボタンをクリックするとすばやく実行に入る．したがって，BASICシミュレーションでよく利用する時間待ちの空ループ：`FOR I = 0 TO 10000 : NEXT I`は，たとえ数10000をもっと大きくしても，あっという間に終わるので，時間待ちには使えない．

そこで，`Riki-0.vbp`で見たように，`Visual Basic`の**Timer機能**を使い，`Timer`の繰り返しの時間（`Timer1.Interval`値）を適当に設定し，`Timer1`プロシージャを実行させている．`Timer1`プロシージャの中で上記の`For ～ Next`文を波の瞬間的描画に使い，この波の伝搬を一定時間ごとの「書き消し法」の繰り返しでシミュレートするプログラムを作成できる（第3章）．

③イベントドリブンとプログラムの構造

すでに述べたようにVisual Basicにはマウスクリックの対象となるコマンドボタンがあり，そのコマンド操作に対してパソコンに応答させるためのプロシージャが対応している（イベントドリブン）．このプロシージャは独立していて，BASICのようにプログラム中の順序で実行されるのではない．極端にいえば，たとえばRiki-0の五つのコマンドボタンの配置順（したがってプログラム内のコマンドプロシージャの順）は，どうでもよいことになる．ユーザーの観点で見やすい配置や順序を決めればよい．後の章で，コマンドボタンの代わりにこれらの命令を画面の上段にメニュー形式に配置する例（最初の例は`Riki-5.vbp`）を示すが，この場合も同様である．

また，ユーザーが`Function`（関数）プロシージャ（例：`Riki-4.vbp`）を定義してその関数記号で他の場所で使ったり，`Riki-0.vbp`のようにZahyoプロシージャを定義し，その`Zahyo`名で他のプロシージャ内に引き移して利用することもできる．しかし，`For ～ Next`の中で`For ～ Next`が使えるように，プロシージャの中に別のプロシージャを含めて使うことはできない．上に述べた波の例のように，プロシージャの中でFor ～ Nextを使うことは多い．

④変数の宣言とデータ型の指定について

プログラムの中で使用する変数は必ず`Dim`で宣言しなければならない．整数は`Integer`型か`Long`型で，小数が含まれる場合は`Single`型か`Double`型で，各変数ごとに指定することが望ましいとされている．しかし本書のプログラムでは多くの変数を使っているので，データ型の指定は省略してある．データ型を宣言しない変数を`Variant`型と言い，メモリを多く要するので好ましくないとされるが，本書で扱うプログラムについてはデータ型を宣言するかしないかでプログラムの実行時間に差が認められなかった．