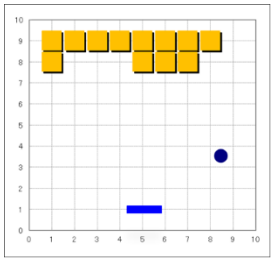
きみろん Comp. 第３章

— ブロック崩しゲームを作る —



はじめに

その昔、コンピュータといえば大学や研究所の計算機センターという建物の中におさまった巨大な図体（ずうたい）の電子計算機を意味していました。しかも、それを使えるのは大学の職員と研究室に入って計算機センターのＩＤパスをもらった学生だけでした。

ところが、1970 年代半ばに革命が起こります。１枚の集積回路にその心臓部分（CPU）が収められ、その巨大な図体が、机の上におけるぐらいの大きさになったのです。

それは、多くの理系人間にとって衝撃的な出来事でした。当時、そのチップで作られたマシンはマイコンと呼ばれ、特に 1977 年アップル社の出した AppleⅡはみんなの憧れの的でした。ソフトは BASIC というプログラミング言語だけで、付録に簡単なゲームがついていました。コンピュータは計算機であると思っていた理系人間にとって、「ゲームができる」ことにその時はじめて気づいた人は多いと思います。その AppleⅡについていたゲームこそ「Little Brick Out」つまり**ブロック崩しゲーム**でした。

この講座では、Excel にある BASIC 言語を使って、**ブロック崩しゲーム**をつくります。まずボールを動かす、そしてボールを壁で反射させる、という具合に一つ一つの動きを理解しながらプログラミングしていきます。

そして、最終的な目標を、この**ブロック崩しゲーム**を参考に「自分のオリジナルゲームを開発する」というところにおくことにします。でもそんなに気張る必要はありません。ちょっと自分のアイデアを使って改良してくれたらいいのです。

結果、君達はプログラミングに必要な考え方を学ぶことになりますが、それ以上に、コードを書いて簡単なゲームをつくったという経験が、コンピューターという道具について改めて考えるきっかけになることでしょう。

以下、山の頂上へ無理なく登っていけるようにこのテキストがガイド役を務めます。途中、分からないところは一緒に山に登っている仲間と解決していってください。それがいろんなことを学ぶきっかけになるからです。解説には Excel2019 を使いましたが、他のバージョンと大きく変わるところはありません。

それでは健闘を祈ります。

STEP 1　準備

１.「開発」タブを加えプロ仕様にする

バージョンが新しい Excel ならはじめから上のリボンのタブの中に「**開発**」タブが出てきているものもありますが、なかったら「**ファイル**」タブをクリックして「**オプション**」の「**リボンのユーザー設定**」を開き「**メインタブ**」の「**開発**」にチェックを入れ「OK」をクリックします。これでエクセルの中に「**開発**」というタブが現れ、そこを開くと、そこはエクセルを動かすコードを入力できる道具がそろった場所になっています。どうですか、エクセルの画面の上のメインタブに「**開発**」というタブが加わっていますか。

２．表の座標を数字にする。

ここでは、まずプログラムコードが書きやすいように Excel のほうの準備をしておきます。今度はエクセルの表の左端を見てください。縦は数字が打ってありますね。上端をみると横にA,B,C･･･になっています。これは普通に使うときは問題がないのですが、プログラムを書こうとするとちょっと不便です。横も数字になっている方が使いやすいのです。

画面上のリボンの「**ファイルタブ**」の「**オプション**」→「**数式**」を選び、**☑ R1C1 参照形式を使用する**にチェックを入れておきます。

R1C1 形式に対して通常の形式を **A1 形式**といいます。R1C1 形式にすると Excel の横の欄がアルファベットから数字になり、セルの場所を(x,y) 座標として表すことができ、より数学の感覚に近くなります。ただ**セル(縦の番号、横の番号)**となります。

３．セキュリティチェックを外す

プログラムを書くと言うことは、コンピューターの心臓部を動かすということです。コンピューターはその心臓部に入り込まれないようセキュリティシステムが構築されています。従って、前もってそのコンピューターの安全装置を外してやる必要があります。「**開発**」というメニューから「**マクロのセキュリティ**」を開き、**◉すべてのマクロを有効にする** をオンにしておきます。また、**開発者向けのマクロ設定**の **☑VBA プロジェクト オブジェクト モデルへのアクセスを信頼する**　にチェックを入れます。これで大丈夫なはずです。

４．「Excel マクロ有効ブック」で保存

始める前に、君の USB の中に「**ブロック崩し**」というフォルダを作りその中に「**ブロック崩し（テキストバージョン)**」という名前のエクセルファイルを入れましょう。

このとき**ファイルの種類**を「**Excel マクロ有効ブック**」として保存します。大変重要ですので次ページに詳しく書きました。

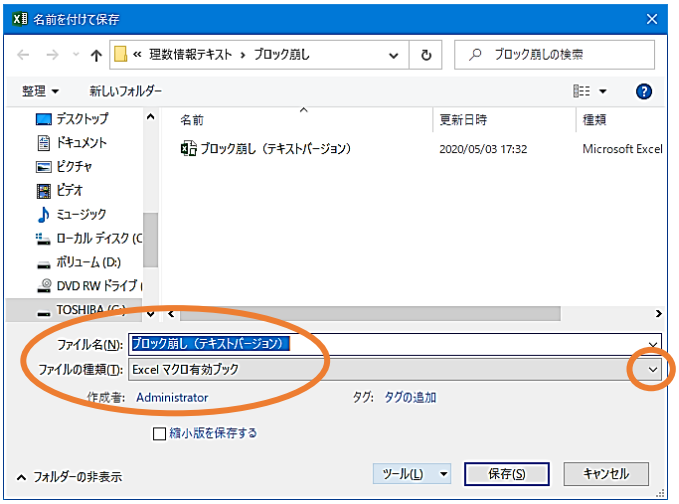


図 3-1-a

図 3-1-b

超重要設定

マクロ有効ブック

この「**Excel マクロ有効ブック**」で保存するという操作は、この「ブロック崩しを作る」講座だけではなく、プログラムを書くファイルすべてに必要です。**大変重要**なので忘れないようにしておきましょう。

操作確認

① 君の USB の「**理数情報**」フォルダの中に「**ブロック崩し**」のフォルダを作っておきます。忘れた人は Comp.1 を見てください。

② 今広げている Excel の「**ファイル**」タブを開き「**名前を付けて保存**」を選択。

③ 「**ブロック崩し**」フォルダの中に**ファイル名「ブロック崩し（テキストバージョン）」**と書く。

④ その下の「**ファイルの種類**」は右にある  をクリックして**図 3-1-b 上**のメニューから「**マクロ有効ブック**」を選択し「**保存**」ボタンをクリック。

「**Excel マクロ有効ブック**」という「**ファイルの種類**」にすることを忘れないようにしてください。というのも、この 1 年間の講座全体で一番多いミスは、せっかくプログラムコードを書いたのに、それを「**Excel マクロ有効ブック**」として保存しなかったばっかりに、次にファイルを開くとプログラムがなくなっている、という事故なのです。

「**まずマクロ有効ブックで保存する**」習慣は、失敗を防ぎ、多くの創造に充てる時間を君に与えます。

マクロの由来：Excel には VBA（ビジュアルベーシックアプリケーション）という BASIC 言語でプログラムを書く機能があってプログラムを学ぶのに適したアプリケーションです。ただプログラムを自在に書ける人はそんなにいないため、Excel には自動でプログラムを書いてくれる機能も備わっています。それが「マクロ機能」といわれるものです。ただプログラミングをする側からすると、余計な機能です。間違って「マクロ機能」のほうに行ってしまう人がいないよう、テキストの適切なところで指示を書きました。

STEP 2　ブロック崩しの現場づくり

作業１　基本データを入力する

ブロック崩しの Excel 側をまず作ってみます。最初は下図の配置をまるごと同じように作りましょう。同じ場所に作ること。数値のセル位置が違うと、プログラムコードがこの例とずれてしまい厄介なことになります。

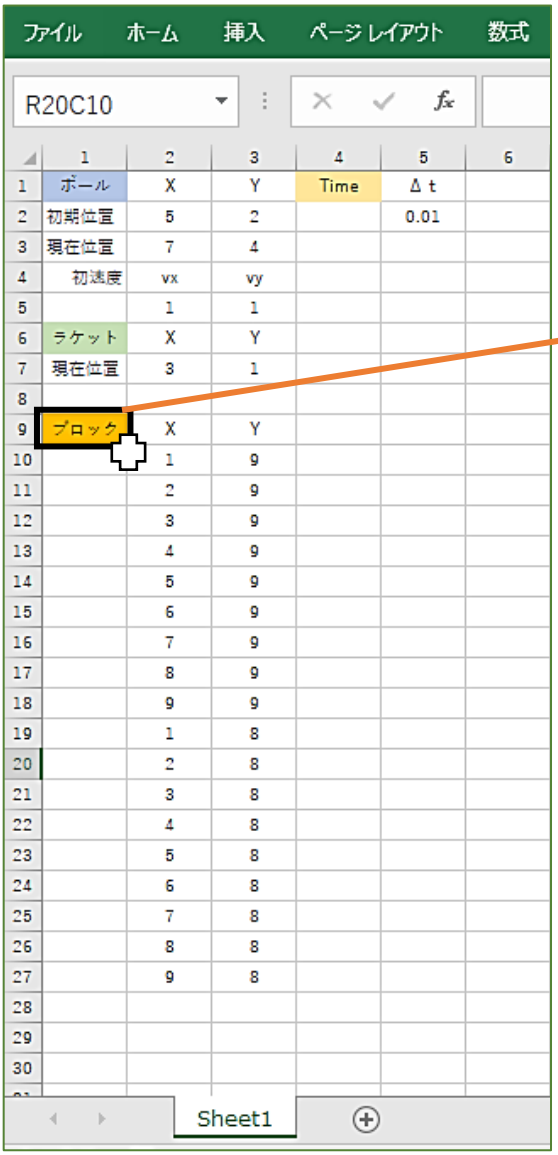


図 3-2

セルの色：セルの色は自由につけることができます。この場合「**ボール**」や「**ラケット**」「**ブロック**」「Time」はこのゲームの大事な要素ですから分かりやすく色を付けておきましょう。色を付けたいセルをクリックし（白十字）「**ホーム**」タブから下の**図 3-2-a** のようにペンキのアイコンを選んで色を決めます。

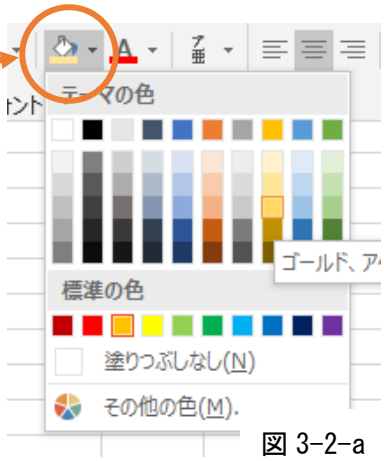


図 3-2-a

セルの文字の中央揃え：セルの中の文字も左右・真ん中などに配置することができます。これも「**ホーム**」タブから**図 2-b** のようなアイコンを選んで「**中央揃え**」をすることができます。

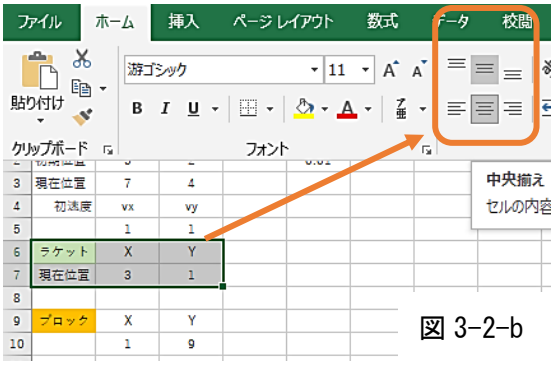


図 3-2-b

	ブロック	X	Y
10		1	9
11		2	9
12		3	9
13		4	9
14		5	9
15		6	9
16		7	9
17		8	9
18		9	9
19		1	8
20		2	8
21		3	8
22		4	8
23		5	8
24		6	8
25		7	8
26		8	8
27		9	8

図 3-3

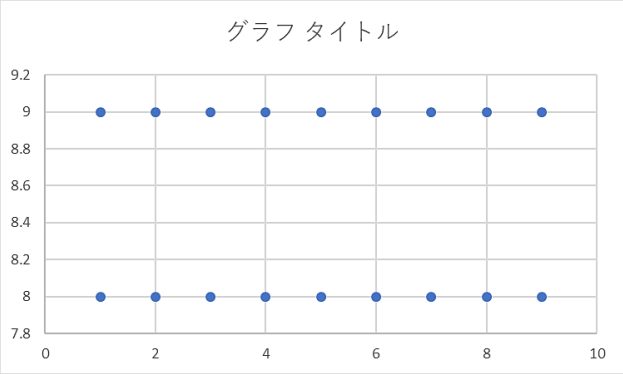


図 3-4

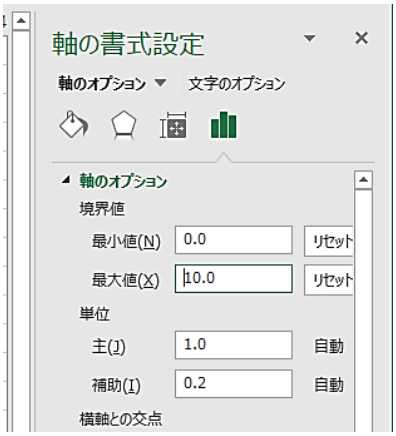


図 3-5

作業２　グラフを挿入する

数値や文字を打ち込んだら、グラフを作ります。このグラフがブロック崩しの舞台になります。

グラフは**散布図**という形式を使って、ボールの位置（X , Y）に丸いボール、ラケットの位置に横長の長方形、ブロック位置に 18 個のブロックが正方形の形で示されているようになっています。**散布図**は、例えばボールの座標を時間ごとに変化させると、ボールが動き出しゲームに最適ってわけです。

まず、ブロック 18 個の**散布図**を作ってみます。ブロック位置の X と Y の列をまとめてドラッグして、**図 3-3** のような状態にして「**挿入**」タグから**散布図**の「**マーカーのみ**」（点だけ）を選びます。

すると**図 3-4** の様なグラフが出ています。でもちょっとがっかりですね。こちらが想定している表紙のようなグラフとはかなり異なっています。しかし、心配することはありません。ここからこのグラフをデザインしていきます。基本的に、このグラフの変えたいところにカーソルを持って行って、右クリックしてメニューを選んでいく方法です。早速やってみましょう。

作業３　グラフを改良する

①　縦軸の目盛りを 0 から 10 までに固定する

縦軸の目盛数字の所にカーソルを持って行って左クリック。目盛がアクティブになったら（四角で囲まれた状態）、右クリックして「**軸の書式設定**」を選びます。**図 3-5** のように「**軸のオプション**」の中で**境界値**の**最小値**を 0.0 にします。また**最大値**は 10.0 にします。これで固定されました。**単位**は主 1.0 補助 0.2 のまま自動にしておきます。

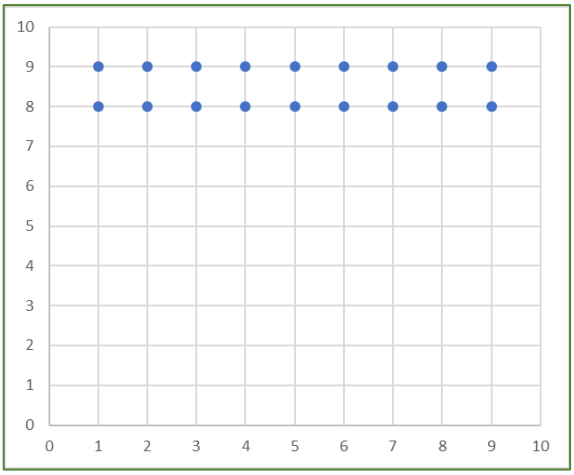
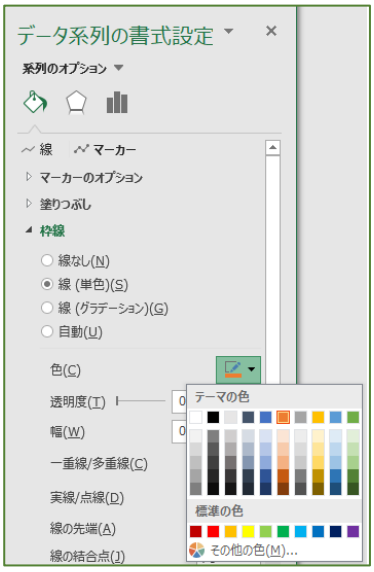


図 3-6

3-7-a

3-7-b



3-7-c

図 3-7 abc

作業３　続き

②　横軸の目盛を固定

次に横軸の目盛数字の所にカーソルを持っていき、①と同じようにしていきます。横軸の場合は初めから**境界値**が 0.0 と 10 になっていると思います。固定したいので 0.01 とかして「**リセット**」にして再び 1 を消して 0.0 とすると「**リセット**」になって固定できます。10 も同様に固定しておきます。「**単位**」のほうも主 1.0 補助 0.2 に固定します。これでボールがどこに行っても舞台は変化しません。

③　「タイトル」を消去

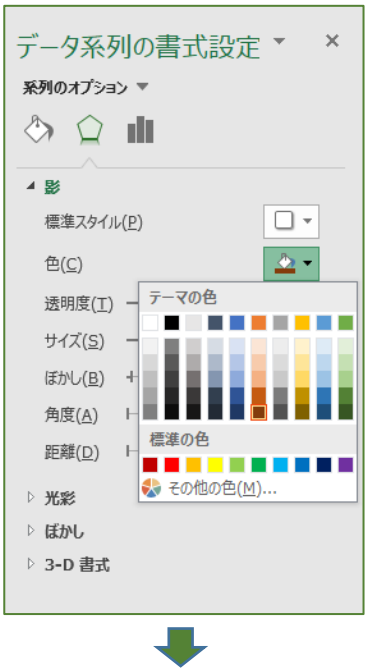
カーソルを「**タイトル**」の所に持っていき右クリック。アクティブになったら左クリックして「**削除**」を選択。

④　グラフの形を正方形に調整

グラフ全体の**外枠**にカーソルを持ってくると、⇔ や ⇕ のような矢印が出てきます。そこをそのままドラッグすると、グラフを正方形の形に持っていけます。ここまでで**図 3-6** のようになります。だいぶ近づきましたが、ブロックがそれらしくありません。もう少しブロックらしくしてみましょう。

⑤　ブロックの変形

ブロックの一つにカーソルを持って行って左クリック。ブロックの周りが泡を吹いたみたいにアクティブになります。そうしたら右クリックで「**データ系列の書式設定**」を選ぶと**図 3-7 a** のメニューが右に出てきます。グラフ上の点のことを「**マーカー**」と呼んでいます。「**缶からペンキが出てるマーク**」をクリックしてそこにある「**マーカー**」を左クリックして「**マーカーのオプション**」「**塗りつぶし**」「**枠線**」というメニューが出てくるようにします。そのメニューをそれぞれクリックしていくと**図 3-7 bc** のように詳細なメニューが出てきます。



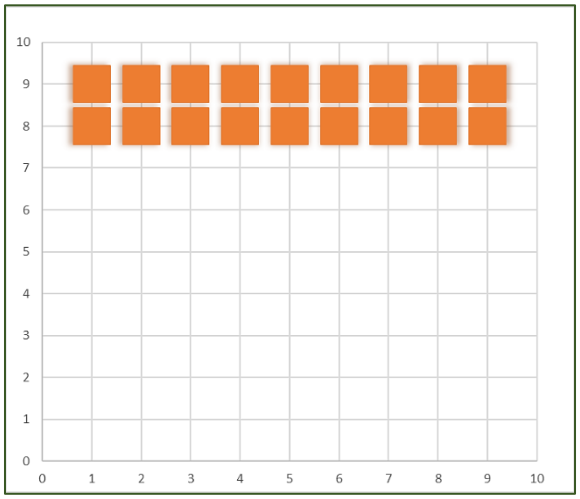


図 3-8

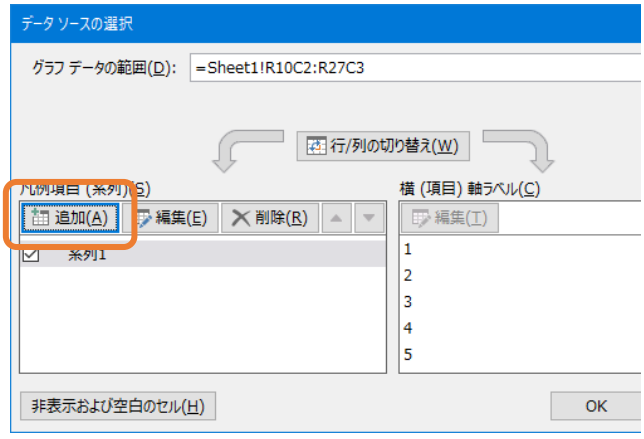


図 3-9

まず「**マーカーのオプション**」のメニューで「**自動**」になっているのを「**組み込み**」にして「**種類**」を■にして「**サイズ**」を 24 にします。ここでブロックがくっついてしまうようなら、グラフそのものの大きさを大きくします。グラフの角にカーソルを持ってきて、カーソルを斜め矢印の形にして引っ張り大きくしてください。角を引っ張ると縦横比は変化しません。

次に「**塗りつぶし**」で「**塗りつぶし（単色）**」を選び、色をテーマの色のメニューの一番上に並んでいる色からレンガの色を選びます。

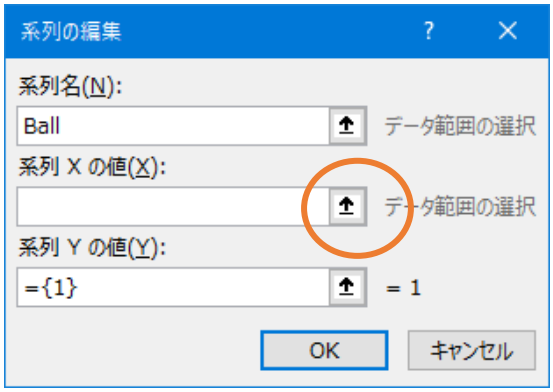
そして次に「**枠線**」のメニューから「**線（単色）**」を選び、やはり色はレンガ色を選びます。

次にこのレンガに影をつけて少し立体的にしておきましょう。**図 3-8** 上の「**系列のオプション**」というペンキ缶のあるメニューの真ん中は、五角形の図形の手前に影がついたようになっていますね。(わかりにくい) これが光を付け加えるメニューです。「影」の標準スタイルからうまく表現できるような「影」を選んで色はレンガ色にしました。(影の形・色は自由です)

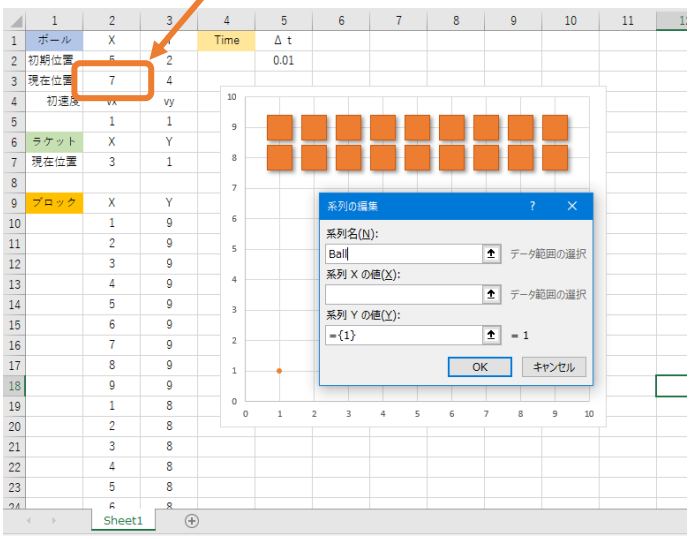
⑥　ボールを登場させる

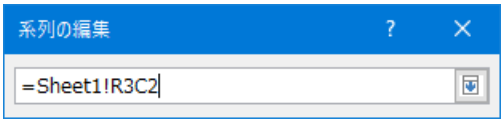
それではこの舞台上にボールを登場させましょう。グラフの端の空白あたりを左クリックしグラフをアクティブ（周りが白丸の泡がついたようになる）にした後、右クリック。「**データの選択**」もしくは「**データ ソースの選択**」のダイアグラムの中の「**追加**」を選んで**図 3-9** のような「**データ ソースの選択**」のダイアグラムを出します。**図 3-9** の囲った「**追加**」をクリックして次のページの**図 3-10** を出してください。

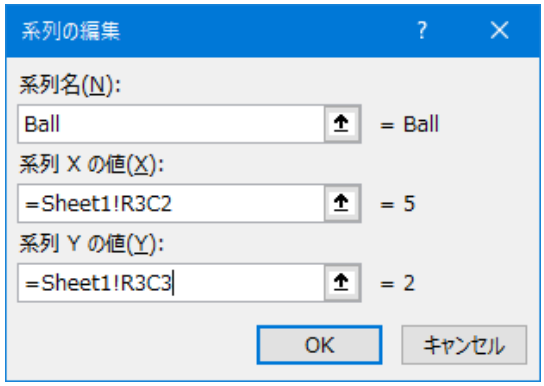
そして「**系列名**」を「Ball」として「**系列 X の値**」の右の↑ をクリックして Excel シートのセル（3，2）を左クリックします。

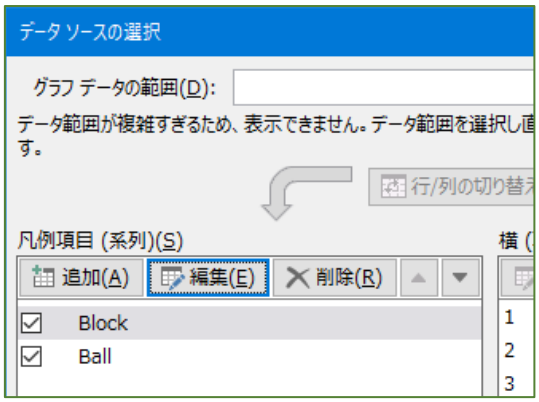


このボールの現在位置の X 座標のセル（3 ,2）を左クリックすると「系列の編集」の中に値が入る。









「系列 Y の値」の↑をクリックして、ボールの現在位置の Y 座標のセル(3,3)を左クリックすると「系列の編集」の中に値が入る。OK をクリック。ついでに系列１の名を「編集」を使って Block としておく。

図 3-10

すると「**系列 X の値**」が

=Sheet1!R3C2

となります。これは「シート１のセル（縦３、横２）の値です」という意味。

「**系列 Y の値**」も同様に入れ**図 3-10** の上から 3 番目のようになれば OK を押して、一番下のようになります。

⑥　系列名 Block

「**データ ソースの選択**」のダイアグラムは**図 3-10** の一番下のようになりますが、「**系列１**」というのは何だったでしょうか。そう、最初に作ったブロック群のことでした。ここでわかりやすくするために名前を付けておきましょう。

「**系列１**」を選び、「**編集**」をクリック。「**系列名**」を Block にして OK を押します。すると**図 3-11** のようになるはずです。

これでわかりやすくなりました。

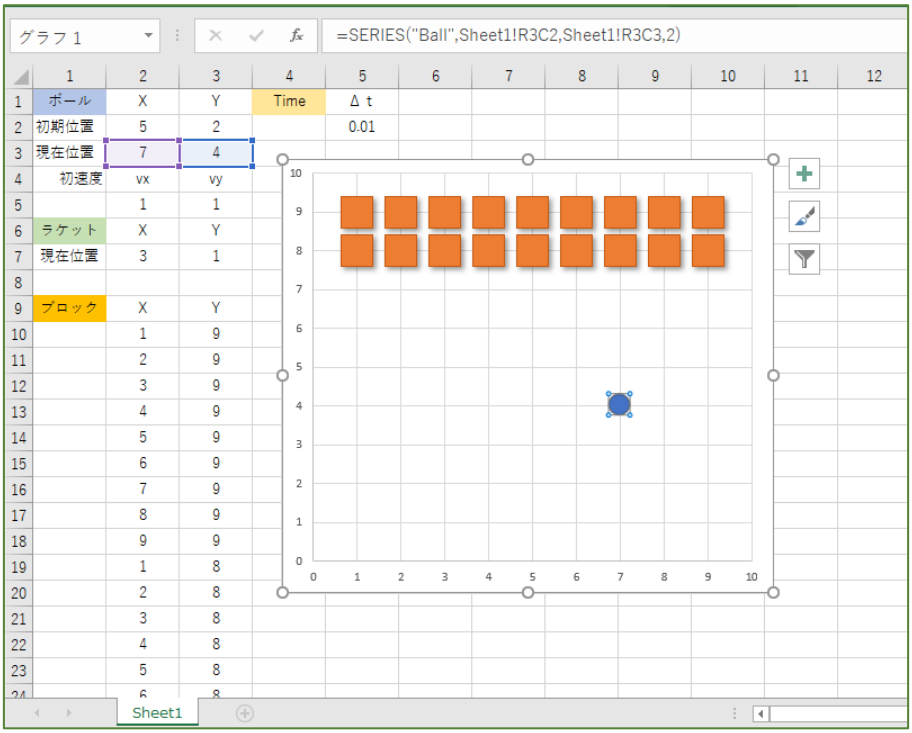


図 3-11

⑨　ボールの形をボールらしくする

これはブロックの形や色を変えた時と同じようにします。ただ、ボールの大きさは**サイズ 16** 程度にしておきます。やたらバカでかいヤツとかやりたくなる人もいるでしょうが、はじめ学ぶときは素直に。というのは、プログラムを組む時、この**ボールやブロックのサイズが大変重要**になってきます。

問題 3-1

図 12 を参考に自分で「ラケット」をグラフの中におき系列名を Racket とせよ。
ラケットの長さは 1.5 目盛程度でラケットの色は自由。これから開発する「ブロック崩し」のプログラムでは、このラケットは左右しか動かない。

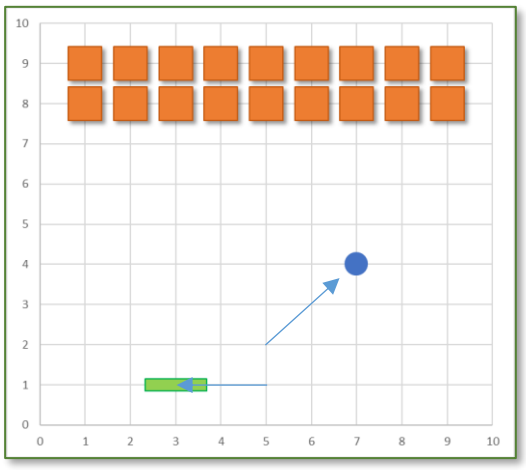


図 3-12

ラケットもグラフの中に挿入すると**図 3-12** のようなデザインになります。

この状態で、ボールの初期位置 X, Y の座標を手動で変えるとボールが動くことを確認しましょう。ラケットも同様です。ラケットの場合は X 座標だけを変えてみます。**図 3-12** ではボールの座標を(5,2)から(7,4) にしています。またラケットの座標は(5,1)から(3,1) にしました。セルの値を変えると、ちょっと動いたように見えますね。そう！ボールを動かすにはセルの値を連続的に変えられるようにすればいいのです！

STEP 3　ボールが動くプログラムを書く

1．プログラムコードを書く白い紙を出す

「**開発**」というメニューから「Visual Basic」を開き、**図 3-13** のような**エディター(プログラム作成編集)**のウインドウを出します。ここにはプログラムコードを書く白い紙があります。白い紙が出ていない場合は、**図 3-13** のようにエディターウインドウの「**表示**」タブから一番上の「**コード**」をクリックします。コードを書いていく実際の作業では、この Visual Basic のウインドウと Excel のウインドウを並べて配置すると作業が格段にしやすいです。

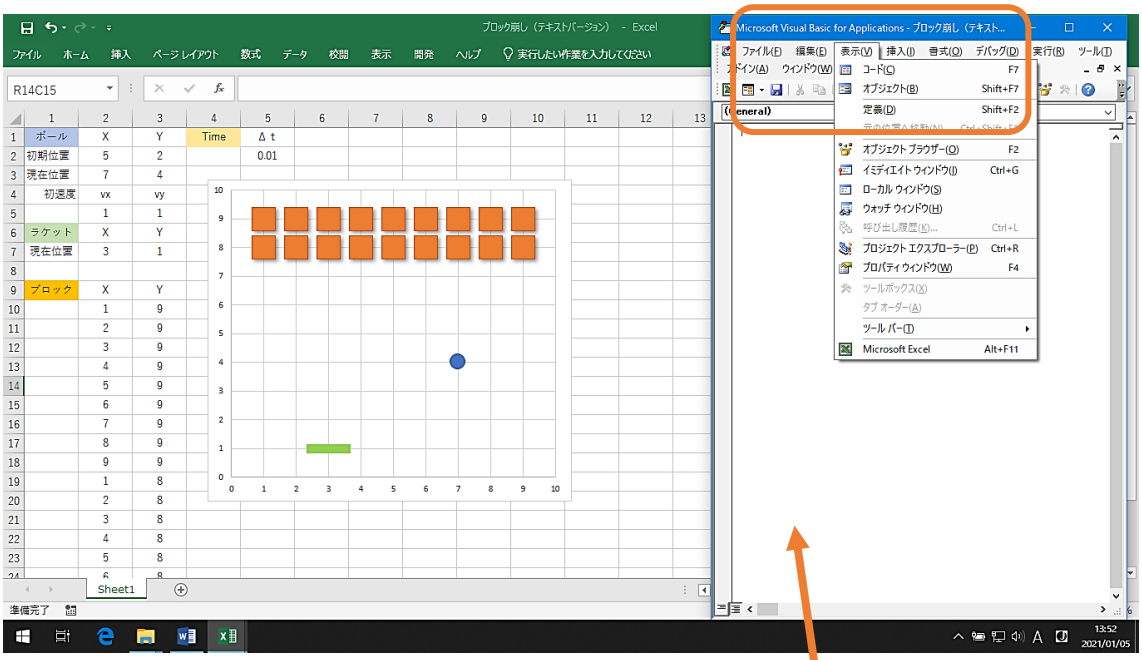


図 3-13

アイコンの意味：**マクロ有効ブック**のアイコンが上のようになっている意味が分かりましたね。アイコンの右下についているトイレットペーパーみたいな紙は、図 13 右のプログラムを書くエディターを意味しているのです。

プログラム言語：ここにプログラムコードを書いていきますが、初め何が書いてあるのかわからないと思います。「**機械と話をする言語**」がプログラムです。いろいろな言語がありますが、まず一つできるようになるとほかの言語を習得する時間は十分の一ぐらいでできるといわれています。BASIC 言語はその名の通り言語学習の基本となる言語です。BASIC 言語には「日本語訳」を付けますので、少しずつ慣れていってください。

最初の２行　宣言文

Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

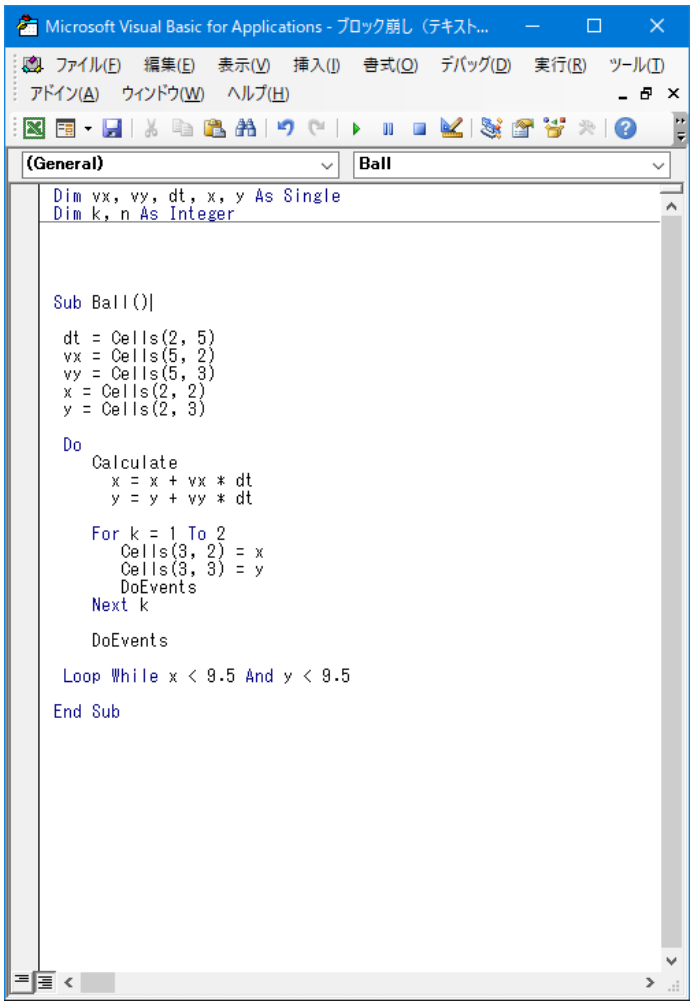


図 3-14

Dim　：Dimension（ディメンション）の略「**次元**」という意味です。

Single：単精度浮動小数点型という「**実数**」です。この倍の精度で計算する方法もあります。これを Double と言います。As は英語の「～として」の意味

Integer:「**整数**」という意味です。

変数は勝手に作れますが、その変数がどんな数値になっているかをはじめに「**宣言**」する必要があります。そのためこの初めの２行を「**宣言文**」と言ったりします。

２．最初の 2 行を書く

半角で最初の２行を書いてみます。書く前にコードを正確に書くコツを伝授します。

書くのは

Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

ですが、大文字と小文字が混在していますね。このとき大事なのは **Dim** と書かずに **dim** とすべて小文字で書くことです。ただし半角空ける所は空けないといけません。つまり

dim vx,vy,dt,x,y as single
dim 半角スペース vx,vy,dt,x,y 半角スペース as 半角スペース single

と書きます。1 行書いたら Enter キーを押します。すると、大文字にすべき所が自然に大文字になります。綴りが正確でコンピューターが知っている言葉だったら大文字になってくれるのです。これで、綴りのミスを防ぐことができます。これだけでも自分のパソコンと会話している感じになります。

それでは最初の２行を書いてみてください。下にどんな意味なのかを書きました。

Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

日本語訳
プログラムの中の変数に使う文字の次元を定義します。
vx，vy，dt, x，y というのは単精度の実数(single)として扱います。K，n は整数(integer)として扱います。

```
Sub Ball()

  dt = Cells(2, 5)
  vx = Cells(5, 2)
  vy = Cells(5, 3)
   x = Cells(2, 2)
   y = Cells(2, 3)

 Do
     Calculate
       x = x + vx * dt
       y = y + vy * dt
     For k = 1 To 2
        Cells(3, 2) = x
        Cells(3, 3) = y
        DoEvents
     Next k
     DoEvents
  Loop While x < 9.5 And y < 9.5

End Sub
```

３．Ball が動く基本プログラムを書く

次にボールを動かす Ball プログラムを書いていきます。左のコードがその Ball という名前のプログラムです。これを見ながらエディターのコードシートにその通り書いていきましょう。

エディターでは

Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

の後に線が引かれていますが、この線は自動で入りますので気にしないでこの次の行から書いていきます。

はじめ

sub Ball()

と書いてエンターキーを押してみてください。

Sub Ball ()
End Sub

といきなり最後の構文も出てきて、大文字になるところは大文字になっています。

Sub というのは VBA ではプログラムのひと塊を「Sub プロシージャ」と呼んでいてその始まりが Sub Ball () で最後の構文が End Sub なのです。この間にプログラムを書いていきます。

半角小文字で書いていくことを忘れないように。式の場合は

dt=cells(2,5)　　x=x+vx*dt

のように半角空けずに書いてもエンターキーを押せば

dt = Cells(2, 5)　　x = x + vx * dt

と勝手に間隔を調整してくれます。

注意しないといけないのはプログラム構文の場合で

Fork=1to2

のように書くとダメです。エラーが出てしまいます。これはプログラム構文の一文ですので

for k=1 to 2

と書きます。エンターキーを押すと

For k = 1 To 2

と大文字、半角空けの所は変換され、正しいプログラム構文なんだと分かってちょっとほっとします。間違ってないってことです。

空白もそのまま真似ましょう。ずいぶん凸凹しているなと感じると思います。なぜこんな書き方をするのでしょう。その理由を次ページに書きました。

３．Ball が動く基本プログラムを書く（続き）

これも理由があってプログラムの構造が分かるように書いています。プログラムを読める人は図 3-15a のようにまず大きく読んでいきます。

プログラムが読める人の読み方例

「Ball プログラムか。End Sub がここだからこの間が Ball プログラムか。（中略）
お、Do Loop の繰り返しがあるぞ。どうやって Loop から脱出するんだろ。Aha!（中略）」

という感じです。すると書き出しが凸凹している意味が分かってきませんか？！プログラムは構造として何回か反復するところが必ずあります。「**反復**」と呼んでいます。（そのまんまです）
今書いている Ball プログラムには「反復」の部分が 2 種類あります。

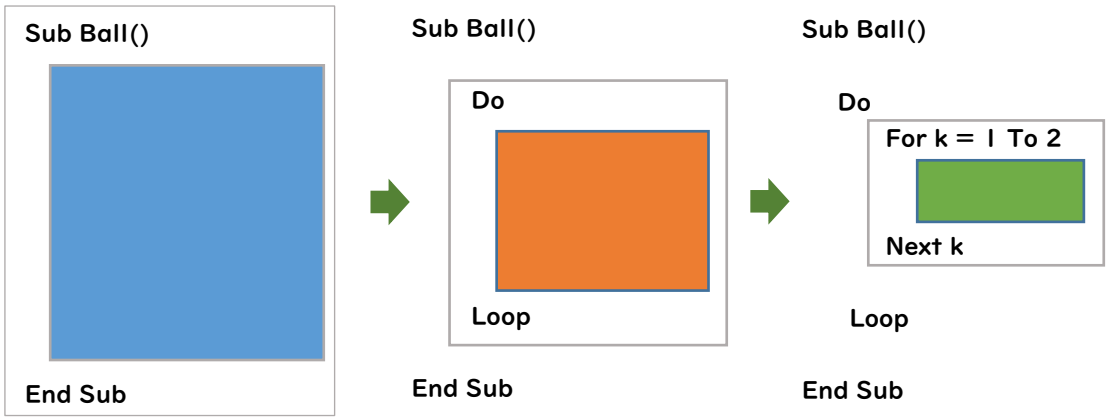
Do Loop 文

For To Next 文

の二つです。その説明を下に書きましたので読んでください。

さて、段落の凹凸のことですが分かりましたね。まずこのような「**反復**」の部分が分かりやすいようにプログラムが書かれてあるということです。皆さんも書くときにはそれに注意して書いていきましょう。なお「**反復**」に対してそのまま上から下へ連続していくプログラム部分を「**連接**」と呼んでいます。

次のページにこのプログラムの日本訳を付けました。読んでみましょう。



Sub プロシージャ

Sub と End Sub に挟まれた部分が Sub プロシージャと呼ばれるプログラム部分。プログラム名は Sub の後につける。今回は名前を Ball とした。() の部分はなぜか必要。

Do Loop 文

Do と Loop で挟まれた部分を「繰り返せ」ということ。このままでは誰か止めてやらないと止まらない。脱出する方法がいくつかある。その方法は図 15b 参照

For To Next 文

For k = 1 To 2 と Next k に挟まれた部分を k の値を 1 つずつ増やしながら k が 2 になるまで繰り返す。
つまり 2 回繰り返した後ここを出て次の命令に行く。k は整数と初めに宣言してある。

図 3-15a

VBA プログラム１

```
Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

Sub Ball()

  dt = Cells(2, 5)
  vx = Cells(5, 2)
  vy = Cells(5, 3)
   x = Cells(2, 2)
   y = Cells(2, 3)


 Do
     Calculate
       x = x + vx * dt
       y = y + vy * dt
     For k = 1 To 2
        Cells(3, 2) = x
        Cells(3, 3) = y
        DoEvents
     Next k
     DoEvents
  Loop While x < 9.5 And y < 9.5

End Sub
```

日本語訳

次の Ball プログラムに使われている変数は
vx, vy, dt, x, y が単精度の実数として
k, n を整数として取り扱います。

このプログラムの名前は Ball です。
まず Excel のシートのセル(2,5)の値を dt とします。同様にセル(5,2)の値を vx ,セル(5,3)の値を vy とします。
セル(2,2)、セル(2,3)の値をそれぞれ x , y とします。
したがって
dt=0.01 , vx=1 , vy=1 , x=5 , y=2
となります。

Do Loop 文

まず計算です。
x に vx・dt を加えたものを新しい x とします。
y に vy・dt を加えたものを新しい y とします。
したがって
X=5+1×0.01=5.01
Y=2+1×0.01=2.01
となります。

次に以下の①と②の作業を念のため 2 回繰り返します。

① 新しい x , y をセル(3,2)とセル(3,3)にそれぞれ入れます。
したがって
セル(3,2)=5.01
セル(3,3)=2.01
になります。

② その x , y の値をボールの座標として
グラフにちゃんと描いてください！
（DoEvents！）

2 回の繰り返しが終わると、もう一度ボールをグラフに描くことをしつこく命令（DoEvents！）（しつこいヤツやなあ）

以上の計算からここまでの作業を
x＜9.5 かつ y<9.5 の間
は繰り返しやり続けてください。
x , y がそれ以外の値になったら終了です。

つまりこのプログラムはボールの座標位置を
0.01 だけ増やしながらボールを移動させ、グラフの外に出かかったら止めるという作業を行わせるプログラムなのです。

Do Loop からの脱出

Do

Loop While 条件式

Do Loop 文

Do と Loop で挟まれた部分を「繰り返せ」という命令だが、このループを脱出するには While を使う方法がある。While の後に脱出する条件を書いておけば、その条件を満たした瞬間にこのループから抜けられる。

図 3-15b

プログラミングで使う不思議な数式

このプログラムの翻訳を読むと、不思議な数式が出てくることが分かります。

```
x = x + vx * dt
y = y + vy * dt
```

この二式は、数学的には変です。大体イコールになるなら vx , vy が 0 か dt が 0 になる必要があります。実は、プログラミングではイコールは違う意味で使います。

日本語訳で書いたように「**x に vx*dt を加えたものを新しい x とする**」という意味なのです。この数学のイコールとの違いを回避するために違う記号で書くプログラミング言語もありますが、ここでは、＝ ではなく ⇐だと認識してください。⇐ がキーボードで入力できなかったために ＝と書くようになったのかもしれません。

n は使わんのかあ！の不思議

このプログラムの最初に n は整数と言っておきながらプログラムの中には出てきません。そんなことがあってもいいのでしょうか。いいんです。作者が n も使うかもしれんと思って最初に勝手に書いておいただけです。つまり、使わんでもいいんですな。

DoEvents の不思議

このプログラムにはしつこく DoEvents という命令が実質 3 回も出てきます。この DoEvents という命令は、Excel のシート上のグラフ作図といった処理を Events と呼んでいてそれをやれ（ Do ）という意味です。でもなかなか一発の命令でグラフをプロットしてくれません。その理由は、計算スピードと作図スピードが大きくずれているためです。最近のパソコンは計算能力が上がってスピードが速く、プログラムで計算した値はすぐにセルの中に現れます。ところが、それをグラフ化する時間は何倍もかかってしまうのです。そのため、この DoEvents 命令がないと両者のタイミングが取れたときだけ作図するようになり、ボールが飛び飛びで移動してしまいます。そのため作図時間を稼ぐため繰り返しを入れてプログラムの速度をわざと遅くしているのです。

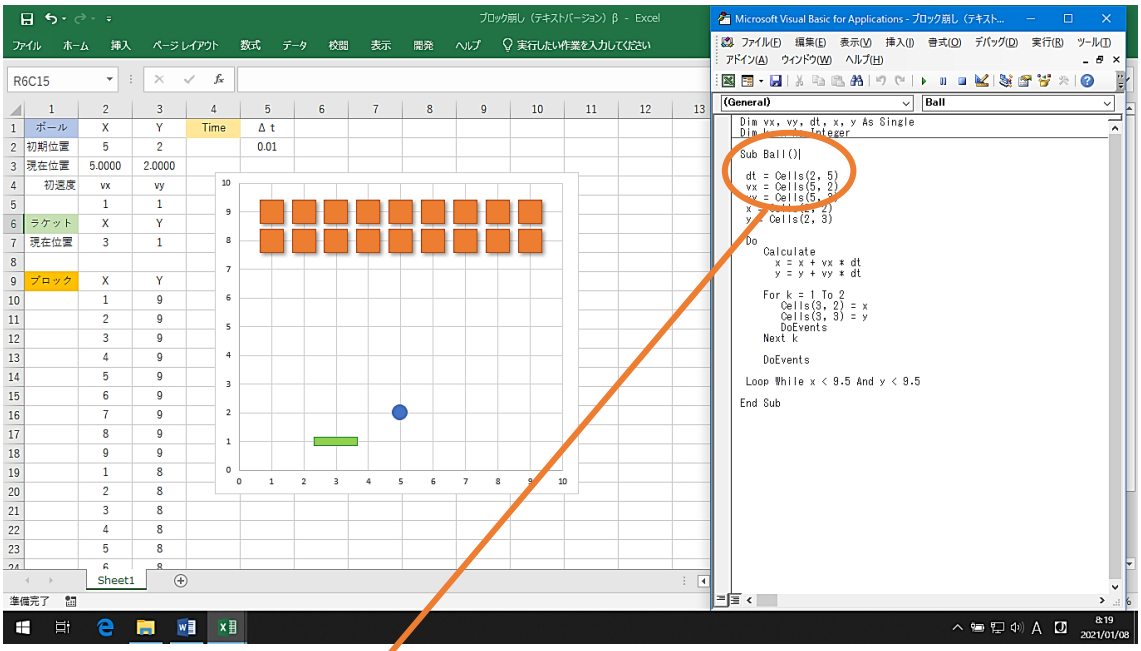
STEP ４　デバッグと実行

１．デバッグとは

デバッグというのはプログラムのミスを発見して修復することです。プログラムは１回書いてミスがゼロで動き出す！ということはなく、どこかに間違いがあります。特にプログラムをやり始めた頃はミスだらけで、ここで「私は向いていない」と思う人と「ミスを見つけてやる」と粘り強く頑張る人に分かれます。一つでもミスを見つけると「小さな幸せ」がそこに待っています。

プログラムコードの中にあるミスのことを「**バグ**(bug)」と呼んでいます。これは「**虫**」という意味でプログラムの「**バグ**」を退治することを「**デバッグ**」と呼ぶわけです。

プログラムエンジニアの世界では、ミスはその作者のせいだけではなく、プログラムという織物を紡ぐ中に現れた虫みたいなものだ、という見方をします。実は大きなプログラムになると、予想もしなかったところに「**バグ**」が現れます。全体の論理と矛盾するところが出てくるのです。つまり、どんなプロフェッショナルも「**バグ**」といつも付き合っているのです。皆さんも、プログラムを学ぶ以上、「**バグ**」は友達みたいなものです。仲良く付き合いましょう。



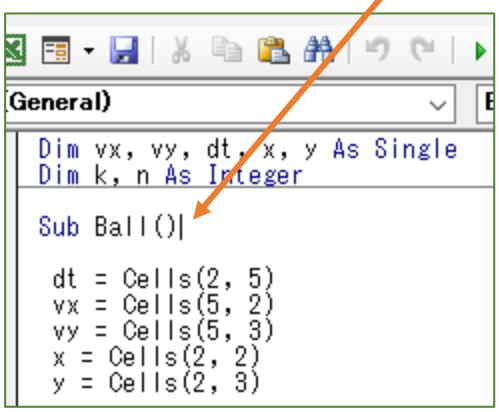


図 3-16

2. デバッグの仕方

プログラムを一応書き終わったら、「**デバッグ**」に入ります。**図 3-16** のように

Sub Ball ()

の右側をクリックして

Sub Ball ()|

とカーソルの縦線が入りチカチカするようにします。

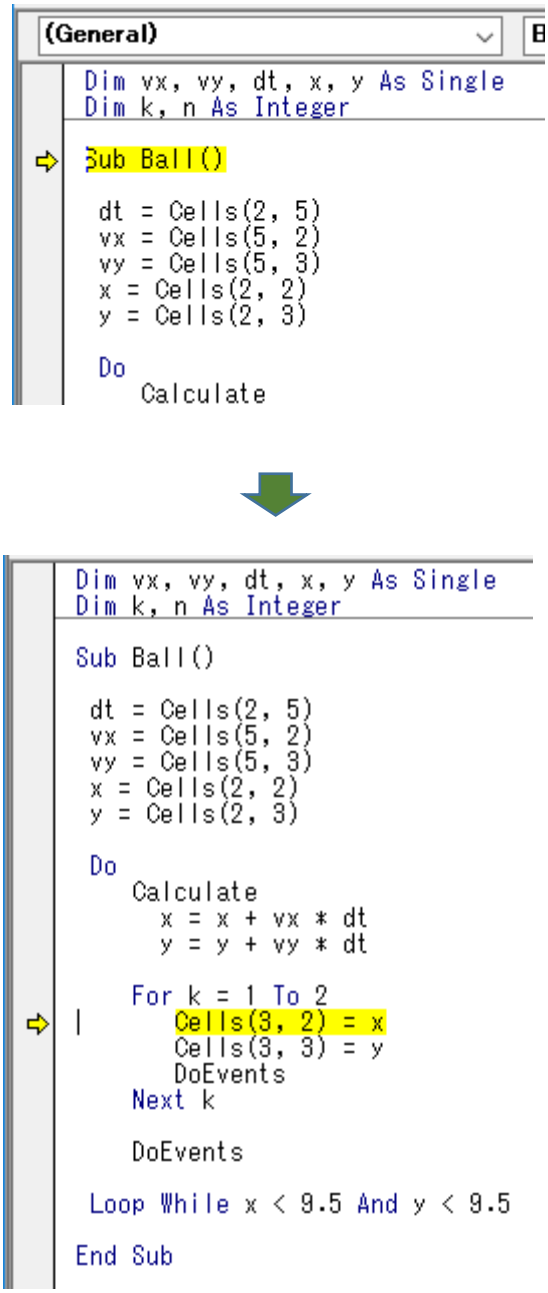


図 3-17

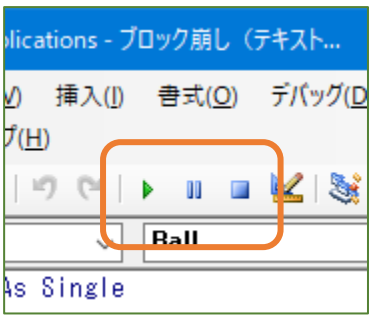


図 3-18

2. デバッグの仕方（続き）

ここで F8 を押すと図 3-17 上のように

➪ Sub Ball ()

黄色の背景になり左に矢印が出てきます。さらに F8 を押すと一行ずつ下がっていきます。これは、コンピューターがプログラムのおかしいところはないか**デバッグ**してくれているのです。

図 3-17 下のように

```
For k=1 To 2
    Cells(3,2)=x
    Cells(3,3)=y
    DoEvents
Next k
```

の所に来たら、ここを 2 回繰り返すか確認してみてください。また、

Do ～ Loop

の間も繰り返すか確認しましょう。

どこにもバグがなかったら Do ～ Loop 間を繰り返していくはずです。

もし**バグ**があって何かメッセージが出てくるようだと、どこかのコードがおかしいということになります。頑張って見つけ出し修正しましょう。

3. プログラムを走らせる

このように F8 を使ってプログラムコードを一行ずつチェックしていくことを「**デバッグする**」ということにしましょう。

バグがなくなったようだったらいよいよプログラムを走らせて、ボールが動いてくれるかを確かめます。エディターウインドウの上のメニューにある図 3-18 の線で囲ったところを見てください。

▶　実行（スタート）

II　中断

■　リセット

を使います。Sub Ball ()| で右にカーソルが来るようにして実行させてみましょう。うまく Excel のシートのグラフの中のボールが動きますか？うまくいくと最後、グラフの端の所で止まってくれます。最初からするには一回リセット■を押してから実行 ▶を押すようにしてください。

STEP 5　コードの理解とマシンとのマッチング

今一度次のコードを見てください。

```
For k = 1 To 2
  Cells(3, 2) = x
  Cells(3, 3) = y
  DoEvents
Next k
```

このコードは、君のマシンの計算スピードと描画スピードをうまくマッチングさせるために、プログラムによってわざと計算スピードを落としてグラフにボールを描画させるためのコードでした。プログラムが動くようになったら、まずここを調整しましょう。

- □　どうしてもボールが飛び飛びでしか描画されない場合は、
  ```
  For k = 1 To 2
  ```
  の数字 2 を増やして 3,4,…と調整してみてください。
- □　逆にゆっくり過ぎるようなら k=1(意味は無くなりますが)としたり、DoEvents をとってしまう方法もあります。このような時
  ```
  For k = 1 To 2
    Cells(3, 2) = x
    Cells(3, 3) = y
    'DoEvents
  Next k
  ```
  と DoEvents の前に**アポストロフィー（数字の７があるキーを Shift ながら押すと「‘」が出てきます）をつけると、この DoEvents コードはコンピューターは読まずに飛ばしてくれます。**ですから
  ```
  ‘マッチング用コード
    For k = 1 To 2
      Cells(3, 2) = x
      Cells(3, 3) = y
      'DoEvents
    Next k
  ```
  というように日本語の前に「‘」をつけると、コードの中に日本語で注釈をつけることもできます。またデバッグの時、怪しいコードの前に「’」をつけてそれをわざと読まなくするというテクニックもあります。
- □　ボールが動くようになったら Δt を 2 倍、3 倍つまり 0.02,0.03 と少しずつ大きくして、自分のパソコンとの相性を調整していきます。一番自然に動く状態はマシンによって異なります。自分のマシンとのマッチングを調べます。一番スローでも自然に動く値を見つけ出しましょう。
- □　次に vx,vy も 2,3 と少しずつ大きくしてどのぐらいの速さまで自然な動きになるかを調べます。最大値を決めましょう。
- □　また、(vx,vy)=(2,3) といったように速さの **x** 成分と **y** 成分の大きさを変えて、ボースの進む向きが変わってくるのを確認しましょう。

こういった作業は大変重要です。自分のマシンとの相性を決定する作業（**マッチング**）を楽しんでやっているうちに、プログラムコードの理解も進みます。

STEP 6　ボールの反射

それでは次にどうやってボールが壁に当たった時跳ね返させるのかを考えましょう。プログラミングには数学や物理の理解が必要な場合もたくさんあります。この講座でも意外なところに数学や物理の知識が出てきて驚くところがあるかもしれません。さらには高校で学ぶ数学や物理の内容が改めて「考える」ことの基礎になっていることを再認識するかもしれません。

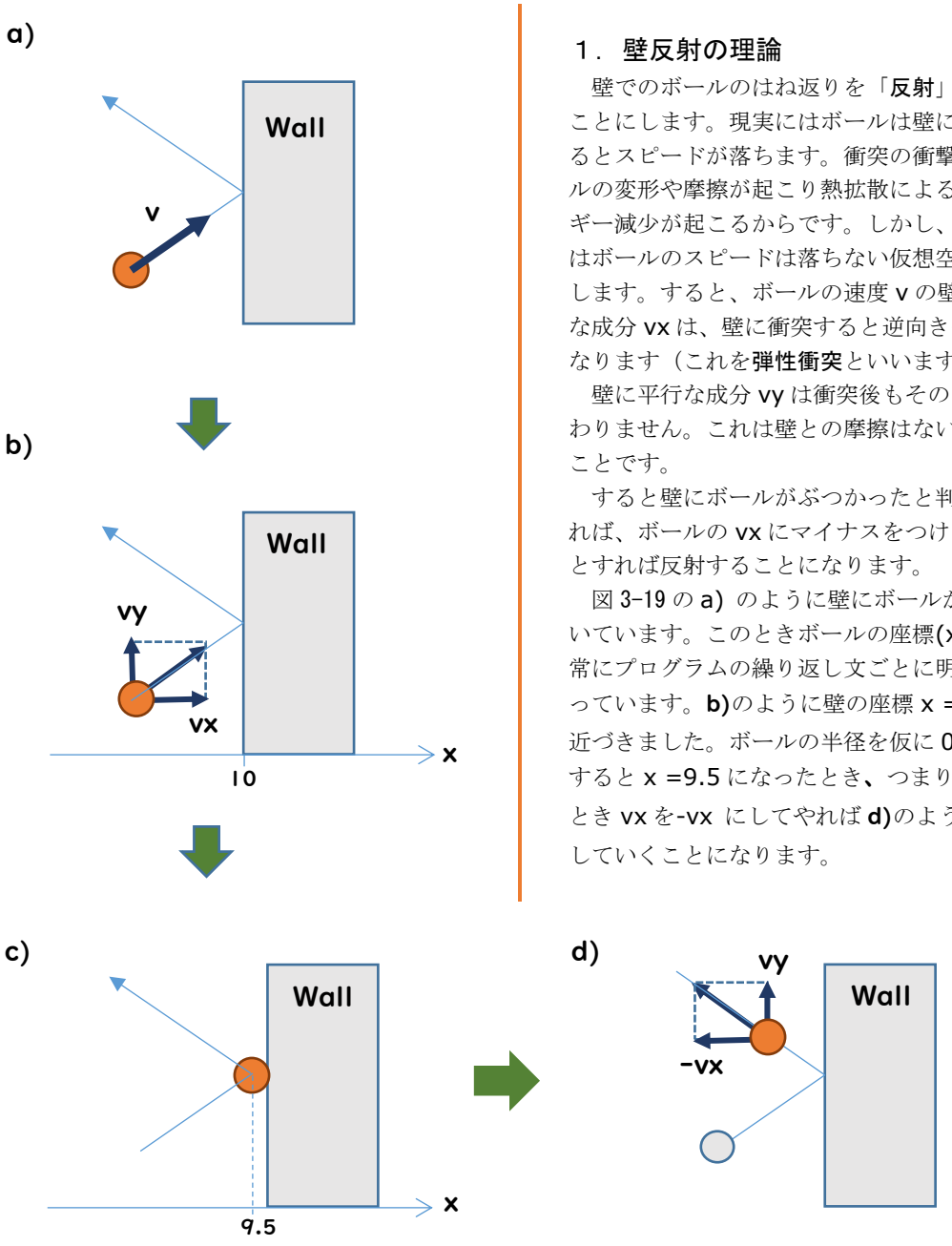


図 3-19

1．壁反射の理論

壁でのボールのはね返りを「**反射**」と呼ぶことにします。現実にはボールは壁に衝突するとスピードが落ちます。衝突の衝撃でボールの変形や摩擦が起こり熱拡散によるエネルギー減少が起こるからです。しかし、ここではボールのスピードは落ちない仮想空間だとします。すると、ボールの速度 v の壁に垂直な成分 vx は、壁に衝突すると逆向き- vx になります（これを**弾性衝突**といいます）。

壁に平行な成分 vy は衝突後もそのまま変わりません。これは壁との摩擦はないということです。

すると壁にボールがぶつかったと判断できれば、ボールの vx にマイナスをつけて -vx とすれば反射することになります。

図 3-19 の a) のように壁にボールが近づいています。このときボールの座標(x,y) は常にプログラムの繰り返し文ごとに明確になっています。**b)**のように壁の座標 x =10 に近づきました。ボールの半径を仮に 0.5 だとすると x =9.5 になったとき、つまり **c)** のとき vx を-vx にしてやれば **d)**のように反射していくことになります。

```
Dim vx, vy, dt, x, y As Single
Dim k, n As Integer

Sub Ball()
    dt = Cells(2, 5)
    vx = Cells(4, 2)
    vy = Cells(4, 3)
    x = Cells(2, 2).Value
    y = Cells(2, 3).Value
    Do
        Calculate
        x = x + vx * dt
        y = y + vy * dt
        For k = 1 To 2
            Cells(3, 2) = x
            Cells(3, 3) = y
            DoEvents
        Next k
    DoEvents
    Call ボールの壁反射
Loop While x < 9.5 And y < 9.5
End Sub
```

a) Value の挿入

b) Call 文の挿入

c) While 文の削除

```
Sub ボールの壁反射()
    Select Case Cells(3, 2).Value
            Case Is >= 9.72
                vx = -1 * vx
            Case Is <= 0.28
                vx = -1 * vx
    End Select
    Select Case Cells(3, 3).Value
            Case Is >= 9.72
                vy = -1 * vy
            Case Is <= 0.28
                vy = -1 * vy
    End Select
End Sub
```

追加の Sub プロシージャ

「ボールの壁反射」

２．ボールの壁反射プログラム

これまでの「**Ball**」というプログラムの下に「**ボールの壁反射**」というプログラムを書きます。プログラムの名前は日本語でも OK です。Excel のエディターのコードシートに **Ball** プログラムの続きとして書いていきます。横線は勝手に入りますので心配いりません。

ただ、これまでの **Ball** プログラムにも変更が出てきます。左の四角で囲んだところをです。それぞれで説明を付けましたので見てください。

a) Value の挿入

これまで

```
x = Cells(2, 2)
y = Cells(2, 3)
```

と書いていたところです。実はそのままでも動くのですが、より詳しく書くと

```
x = Cells(2,2).Value
y = Cells(2,3).Value
```

となります。Value は「値」という意味です。セルには色や値、数値の種類など多くのファクター（要素）がありますので、ここではあえて「値」と書きます。これが「**ボールの壁反射**」プログラムで意味を持ってきます。

b) Call 文の挿入

これは「**ボールの壁反射**」というプログラムを呼び出す（**Call**）命令です。

つまり **Ball** というプログラムを実行すればボールの位置が少し移動した時点でその度ごとに、ボールを反射させるかどうかを判断させるプログラムに行き、その判断と動作を実行してまたこの **Ball** というプログラムに帰ってくるのです。

c) While 文の削除

ボールを壁で反射させるので、この While x < 9.5 And y < 9.5 はいらなくなります。まあ、Do Loop の中でずっとボールは反射し続けることになります。

VBA プログラム 2

```
Sub ボールの壁反射()
    Select Case Cells(3, 2).Value
        Case Is >= 9.72
            vx = -1 * vx
        Case Is <= 0.28
            vx = -1 * vx
    End Select
    Select Case Cells(3, 3).Value
        Case Is >= 9.72
            vy = -1 * vy
        Case Is <= 0.28
            vy = -1 * vy
    End Select
End Sub
```

日本語訳

このプログラムの名前は「**ボールの壁反射**」です。

Select Case 文
セル(3,2)つまりボールの x 座標の値について次のケースの場合の手順に従え
x 座標の値が 9.72 以上のケースでは
ボールの速さの x 成分 vx の符号を逆にせよ。
（vx = -1*vx の=はイコールではなく-vx を改めて vx としなさいということ）
x 座標の値が 0.28 以下のケースでは
ボールの速さの x 成分 vx の符号を逆にせよ。

セル(3,3)つまりボールの y 座標の値について次のケースの場合の手順に従え
y 座標の値が 9.72 以上のケースでは
ボールの速さの y 成分 vy の符号を逆にせよ。
y 座標の値が 0.28 以下のケースでは
ボールの速さの y 成分 vy の符号を逆にせよ。

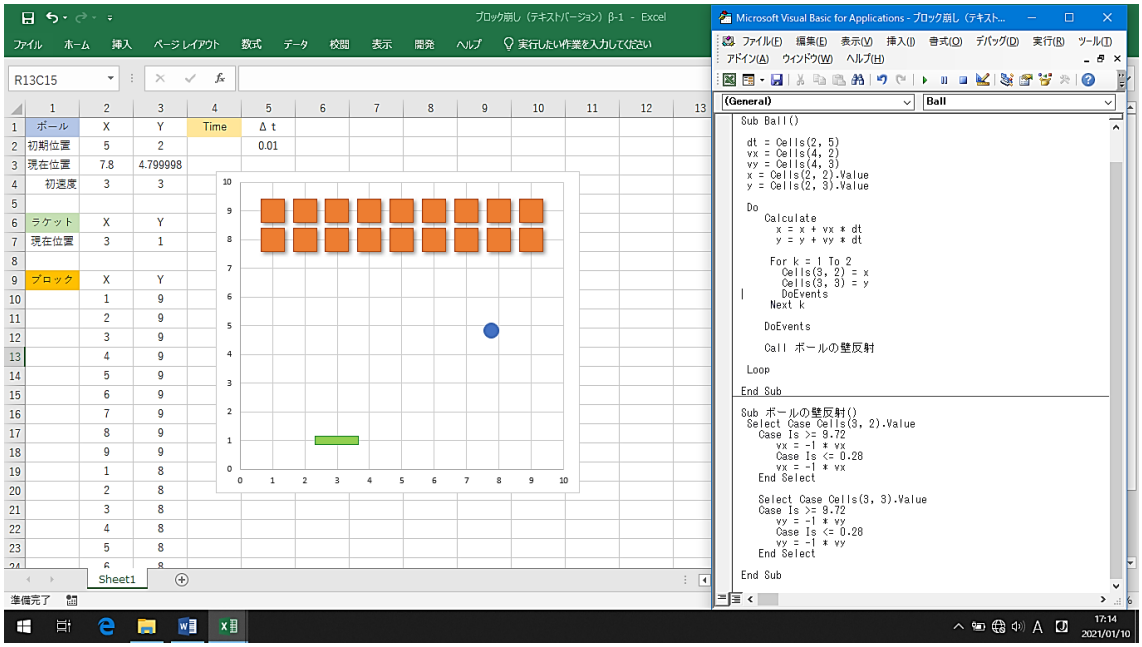


図 3-20

3. Ball プログラムのディバッグと実行

改めて Ball プログラムのディバッグを F8 を使って行ってください。ボールが反射するまでは時間がかかりすぎるので、今度は Ball プログラムの中の Call 文「ボールの壁反射」にきたら、ちゃんと「ボールの壁反射」の方に飛んでまた「Ball」のほうに戻ってくるかどうかがポイントです。
Do Loop 文の中でループしているかも見てください。配置は図 3-20 のようにしておきましょう。

うまくいきそうだったら ▶ を使って実行してください。やり方は STEP 4 で説明した実行方法と同じです。うまく壁で反射するかがポイントです。ドキドキしながら実行させましょう。

STEP 7　ブロック崩し

ボールは 10×10 のエリアの中を反射しながら動き回るようになりましたか。次はいよいよブロック崩しに入ります。ボールがブロックに衝突したらブロックは壊れてなくならなければいけません。そんなことができるのでしょうか。しかもボールはブロックに当たったら反射することになります。どのようにすればいいのでしょう。

	1	2	3
8			
9	ブロック	X	Y
10		1	8
11		2	8
12		3	8
13		4	8
14		5	8
15		6	8
16		7	8
17		8	8
18		9	8
19		1	9
20		2	9
21		3	9
22		4	9
23		5	9
24		6	9
25		7	9
26		8	9
27		9	9
28			

図 3-21

１. ブロック崩しの理論

①ブロックの座標を n で表す

まずブロックは 18 個ありますから
n を 0 から 17 までの整数とすると
ブロックの x 座標が書かれたセルは
　　Cells(10+n , 2)　 n=0,1,2…,17
ブロックの y 座標が書かれたセルは
　　Cells(10+n , 3)　 n=0,1,2…,17
と表すことができます。（図 3-21 参照）

②ブロックの x y 座標を n で表す

ブロックの x y 座標を
　x 座標　b_px(n)　y 座標　b_py(n)
とします。ちょっと面倒な書き方をしてしまいましたが、**Block Position**（**ブロック位置**）の意味で b_p となっています。_ はアンダーバーといい、半角にしてキーボードの「ろ」のキーを Shift キーを押しながら押すと出てきます。

このめんどくさいアンダーバーは意外に使われるので練習のため使ってみました。

すると次のような式が成り立ちます。

b_px(n)=Cells(10+n,2)
b_py(n)=Cells(10+n,3)
　　ただし n=0,1,2…,17

この意味は

ブロック n=5 の
X 座標 b_px(5) はセル（10+5,2）の値
y 座標 b_py(5) はセル（10+5,3）の値
ということになります。

するとブロックが実際はどこに配置されていても n を変えるだけでその座標が読み取れることになります。

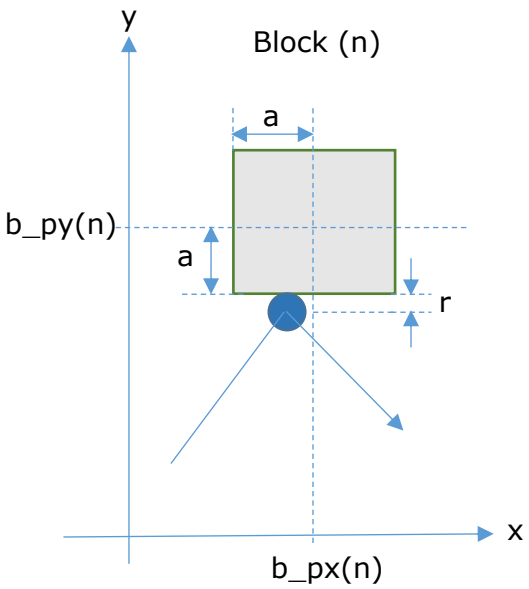


図 3-22

1．ブロック崩しの理論（続き）

③ Block (n)の下面での反射

ブロックの座標を n で一般化することができましたので、次はブロック n のボールの反射を考えます。

図 3-22 のような状態を考えましょう。

するとボールの座標(x , y)が

b_px(n) - a < x < b_px(n) + a

のとき

b_py(n)- a – r < y < b_py(n)- a

の範囲に来たらボールを反射させる

つまり　vy = -1* vy

としてやればよいことが分かります。

これをコード化してやればよいわけです。

④ Block (n)の上面での反射

ブロックの上面でも反射する条件を図 22 を使って考えてみましょう。すると

ボールの座標(x , y)が

b_px(n) - a < x < b_px(n) + a

のとき

b_py(n)+a < y < b_py(n)+a+r

の範囲に来たらボールを反射させる

つまり vy= -1* vy

としてやればよいことが分かります。

⑤ Block (n)の上下面での反射

③と④を合わせて考えると

a + r = d として

ボールの座標(x , y)が

b_px(n) - a < x < b_px(n) + a

のとき

b_py(n)- d < y < b_py(n) + d

にいたらボールを反射させるということでも反射しそうです。これじゃ「ブロックの中に入ってしまうのではないっすか？」そうです、しかし多分反射の判断はブロックの中に入る前に実行されると思いますよ。

⑥ Block (n)の左右面での反射

これまでの式の類推から

ボールの座標(x , y)が

b_py(n) - a < y < b_py(n) + a

のとき

b_px(n)- d < x < b_px(n) + d

にいたらボールを反射させるということで行けそうですね。

```
Dim vx, vy, dt, x, y As Single
Dim k, n As Integer
Dim b_px(100), b_py(100), b_vx(100), b_vy(100) As Single

Sub Ball()
      （省　略）
 Do
      （省　略）
    Call ボールの壁反射
    Call ブロック崩し
 Loop
End Sub

Sub ボールの壁反射()
      （省　略）
End Sub

Sub ブロック崩し()
   For n = 0 To 17
       b_px(n) = Cells(10 + n, 2)
       b_py(n) = Cells(10 + n, 3)
      If y > b_py(n) - 0.55 And y < b_py(n) + 0.55 Then
            If x < b_px(n) + 0.6 And x > b_px(n) - 0.6 Then
                  vx = -1 * vx
                  課題コード
               Else
            End If
         Else
      End If

      If x > b_px(n) - 0.55 And x < b_px(n) + 0.55 Then
            If y < b_py(n) + 0.6 And y > b_py(n) - 0.6 Then
                  vy = -1 * vy
                  課題コード
               Else
            End If
         Else
      End If
   Next n
End Sub
```

a) 宣言文の挿入

b) Call 文の追加

c)プログラムの追加

２．プログラム追加

a) 宣言文の挿入

b_px(100)のような変数を**一次元配列変数**といいます。この場合、単精度の実数であるということですが、n=100 まで使えるということになります。100 個もブロックを使うことはないと思いますが、余裕をもって宣言しておきます。

b) Call 文の追加

ボールの壁反射の判断をした後は、ブロックに衝突したかのプログラムを走らせて方ここに帰ってきます。

c)プログラムの追加

「**ブロック崩し**」というプログラムを追加しました。まずはこの通り書いてみましょう。
前ページで出てきた a と d は a=0.55 , d=0.6 としています。

ここで使われているのは If Then Else 文といわれているものです。

前頁の理論をコード化したものです。詳しくは次ページで解説します。

なお四角で囲んだところは、ボールが衝突したブロックを壊すためのコードが書かれています。これは君が考えて作ってください。まずは、ここを書かなくてもプログラムは走ります。まずは F8 でディバッグして実行してみましょう。

```
Sub ブロック崩し()
    For n = 0 To 17
        b_px(n) = Cells(10 + n, 2)
        b_py(n) = Cells(10 + n, 3)
If y > b_py(n) - 0.55 And y < b_py(n) + 0.55 Then
        If x < b_px(n) + 0.6 And x > b_px(n) - 0.6 Then
            vx = -1 * vx
            課題コード
        Else
        End If
    Else
End If

    If x > b_px(n) - 0.55 And x < b_px(n) + 0.55 Then
        If y < b_py(n) + 0.6 And y > b_py(n) - 0.6 Then
            vy = -1 * vy
            課題コード
        Else
        End If
    Else
    End If
    Next n
End Sub
```

```
If 条件式１  Then
    処理１
      If 条件式２  Then
        処理２
      Else
      End  If
Else
End If
```

もし条件式１を満たすなら処理１をしなさい。それ以外は何もせず終わんなさい。処理１はもし条件式２を満たすなら処理２をしなさい。それ以外は何もせず終わんなさい。

日本語訳

```
If（もし）y が b_py(n)±0.55 の間にあるか

  Then（そうなら）次の処理をしなさい。

    その時 If（もし）x が b_px(n)±0.6 の間にあるか
      Then （そうなら）
          vx の符号を変えたものを vx にして
          ブロックは破壊しなさい。
      Else（それ以外なら）
          なにもせず
      End If　（終わりなさい）

  Else （それ以外なら）何もしなくてよい

  End If　終わりなさい
```

図 3-23

２．「ブロック崩し」のプログラムの特徴

この Sub ブロック崩し() の特徴は、図 3-24 のようにボールが数多くあるブロックに当たったか当たらなかったかを、一般的なブロック n に当たる条件を書き出し、それを For n=0 To 17～Next n で次々と n の値を増やしてブロック一個ずつ実行していくという手法です。

このプログラムは、

IF 文と For To Next 文（反復）

でできています。この **IF 文**を一般的に「**条件分岐**」といいます。先の壁反射で出てきた **Select Case 文**も「**条件分岐**」の一種です。この講座の最後にすべての「**条件分岐**」の構文をまとめたいと思います。

実は、プログラムコードはこの「**条件分岐**」と「**連接**」「**反復**」のたった 3 種類でできています。どんな複雑なコードもこの 3 種類を使って作ることができることが分かっていて、それは驚くべきことです。

これまでにそのエッセンスがすべて出てきました。またシンプルなだけに、うまくアイデアを出すことがどれだけ重要かもわかると思います。

君たちに課された課題、どうやってブロックを破壊するかも、どうやってアイデアを出すかという問題なのです。

プログラムコードはアイデアの宝庫といってもいいでしょう。この講座のプログラムコードもすべてオリジナルなものです。知的財産に対する観点から、プログラムを書く人に敬意を払うことも学んでくれると嬉しいです。

```
Sub ブロック崩し()
  For n = 0 To 17
    n番目の Block n について
    ボールが衝突したかどうかの
    条件分岐が書かれたコード
  Next n
End Sub
```

Δｔだけ進むたびに一般的な Block n についての衝突条件にｎ＝０からｎ＝１７までの値を入れながら繰り返しチェックしていく。コンピューターというのはすごいヤツですな。疲れを知らない。

図 3-24

問題 3-2

図 23 に書かれた「ブロック崩し」のコードの中に「課題コード」と書かれたブランクがある。ここにボールがブロックにぶつかったときブロックが壊れてなくなるという「コード」を自分で作りなさい。また、それを使って全体のプログラムを動かしなさい。

STEP 8　ラケット反射

ボールが壁でもブロックでも反射するようになりましたか。ブロックは初期条件（図 3-21）を変えるとどこにでも配置することができます。ためしにやってみて、ブロックの下側だけではなく、上から右から左から衝突してもちゃんと反射するか、また君の考えたコードでブロックはボールが当たると壊れて消滅してしまうのかやってみてください。衝突面を決めている a や d の値（図 3-22 参照　d=a+r ）もリアルさが出るようにマシンに合わせて少しずつ調整してみてください。

さて、このゲームは参加者がラケットを動かしてブロック崩しにチャレンジするものです。次は、ラケットでボールを受け止めたらボールが反射するようにしてみましょう。

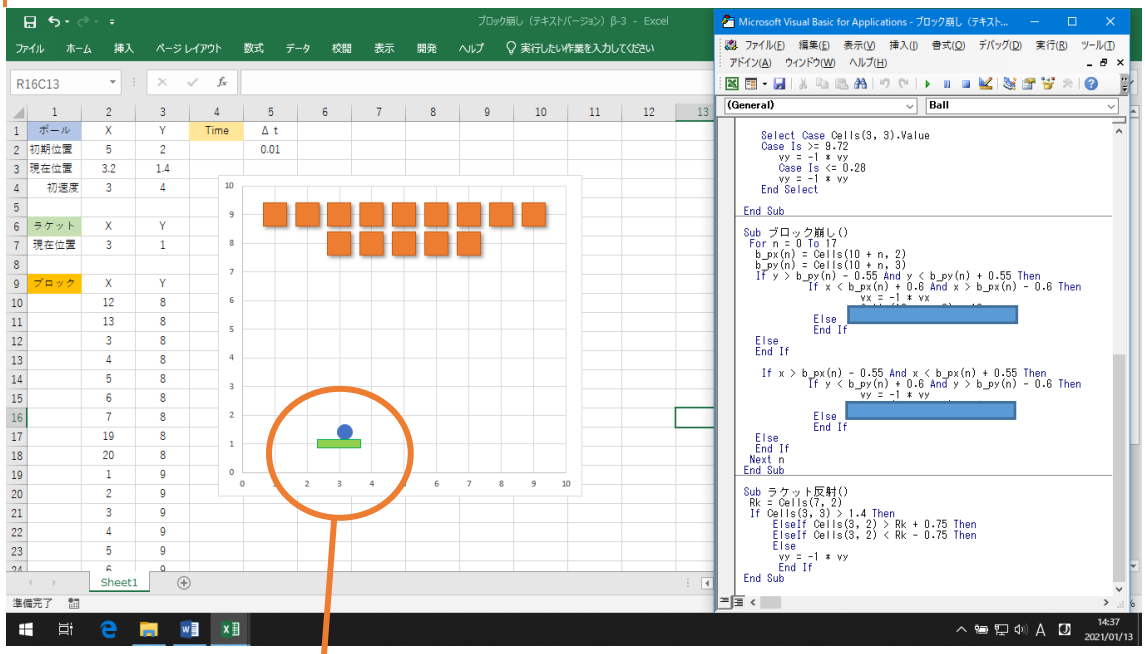


図 25

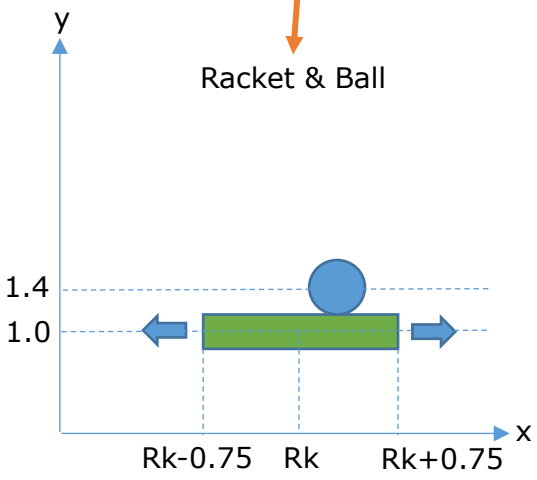


図 3-26

1. ラケット反射の理論

①ボールが反射する領域

ボールの位置の y 座標が

y=1.4 のとき

ラケットの x 座標を Rk として

Rk-0.75 ＜ Rk ＜ Rk+0.75

ならばボールは反射する

つまりボールの速さの y 成分 vy を

vy=-1*vy

とすればよいわけです。これをプログラムコードでどうやって書くか。ちょっと変わった書き方を紹介します。

```
Sub ラケット反射()
    Rk = Cells(7, 2)
      If Cells(3, 3) > 1.4 Then
          ElseIf Cells(3, 2) > Rk + 0.75 Then
          ElseIf Cells(3, 2) < Rk - 0.75 Then
          Else
            vy = -1 * vy
       End If
    End Sub
```

日本語訳

ラケットの x 座標セル(7,2)の値を Rk と置きなさい。
If(もし)ボールの y 座標セル(3,3)の値が 1.4 より大きいなら
Then（そうなら）なにもせず（空白）
ElseIf (それ以外でもし)
　　ボールの x 座標セル(3,2)の値が
　　Rk+0.75 より大きかったら
Then（そうなら）なにもせず
ElseIf (それ以外でもし)
　　ボールの x 座標セル(3,2)の値が
　　Rk-0.75 より小さかったら
Then（そうなら）何もせず
Else(それ以外だったら)
　　vy=-1*vy
　　として vy の符号を変え反射させなさい。

	1	2	3
1	ボール	X	Y
2	初期位置	5	2
3	現在位置	3.2	1.4
4	初速度	3	4
5			
6	ラケット	X	Y
7	現在位置	3	1
8			

図 3-27

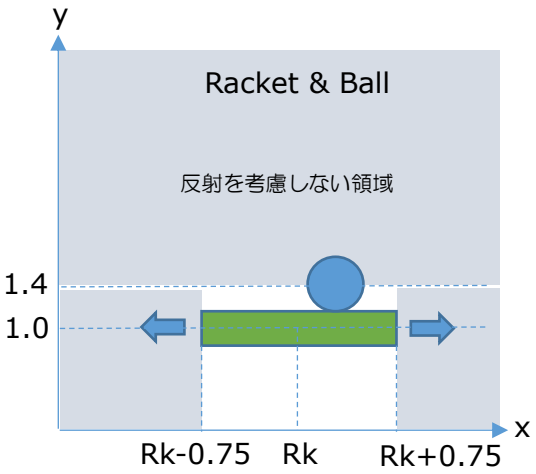


図 3-28

２. ラケット反射のプログラム

左上は If Then ElseIf 文という If 文で書かれているコードです。右側に日本語訳が書かれています。**「ラケット反射」**のコードと合わせて図 3-27 と図 3-28 を見ながら読んでいってください。

分かりましたか？ざっくり言うと
「ボールの y 座標が y >1.4 なら反射する必要はないよね。」
「またそれ以外（つまり y<=1.4）でも x > Rk+0.75 だったり x<Rk-0.75 だったらラケットの左右の外だから反射できないよね。」
「それ以外なら反射だよ。」
といって確実に反射しないエリアを決めるコードになっています。
ただ**図 3-28** を見ると分かるようにラケットの下側は、反射する領域になっています。この辺りはもちろん y=0 の近辺は壁反射をするのでどうなるかは実際にプログラムを走らせて見てみることにしましょう。

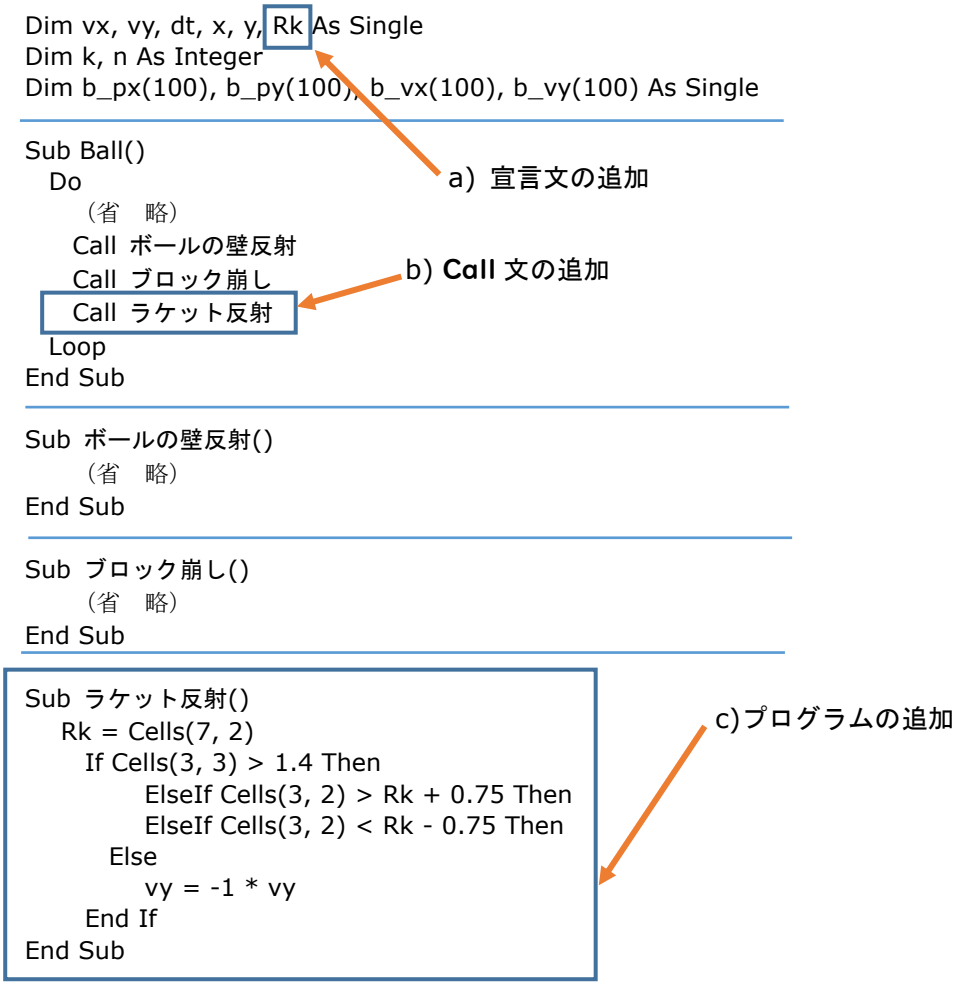


図 3-29

2. 全体プログラム

図 3-29 はこれまでの全体のプログラムです。追加するところを忘れないようにしましょう。

a) 宣言文の追加

ラケットの位置（x 座標）を表す変数 Rk を単精度の実数としておきます。

b) Call 文の追加

Ball プログラムの中にラケット反射を判定するプログラムに行く Call 文を入れておきます。

c) プログラムの追加

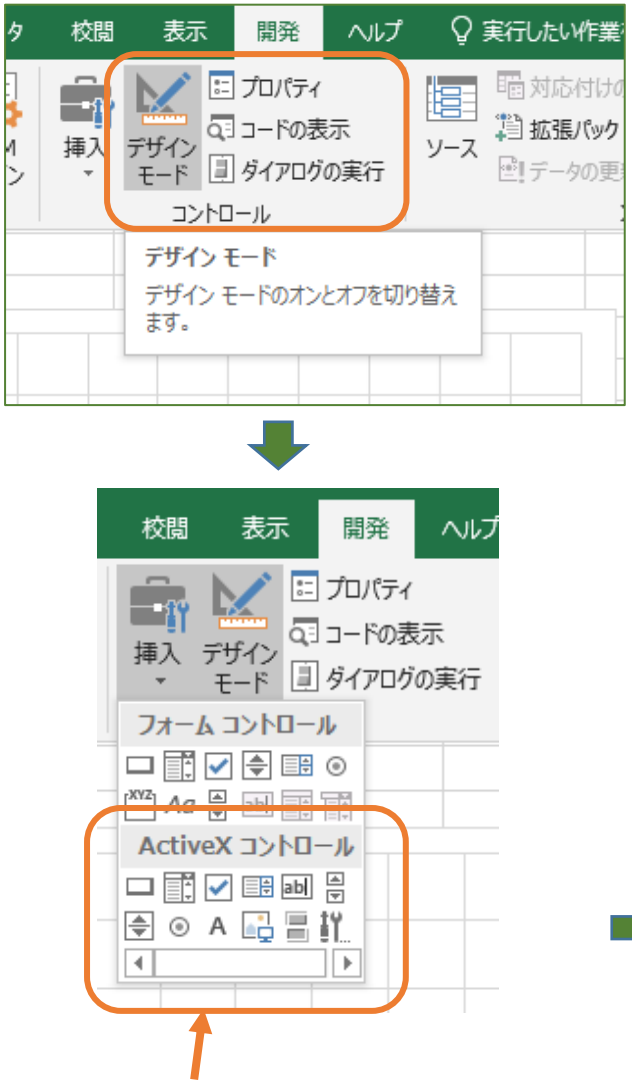
最後に「ラケット反射」のプログラムを書きます。その後 F8 でデバッグした後、実行しラケットでの反射をみます。まだラケットが動かないので、全体の動きはラケットの座標セル(7,2)(7,3)の値を変えて、ラケットを左右に動かして調整してください。

STEP 9　ブロック積みボタンの作成

作業１　ボタンの設定

ブロックをいちいち積み上げるのはいい加減面倒ですね。ボタン一つでブロックが積み上がるようにしましょう。これからの講座でもボタンを作っていく機会が毎回出てきます。このボタンとは、ボタン一つで一つのプログラムをスタートさせたりストップさせたりするもので、大変便利です。

それではブロックを積むボタンを作りましょう。



「ActiveX」群から選ぶこと！
上の「フォームコントロール」のボタン群はプログラミングでは使えない。まちがって上側を選んでプログラムが動かないと悩む人がたくさんいる。

① デザインモードをオン

「**開発**」のタブから「**デザインモード**」をクリックしてオンにします。オンになると色が濃くなります。

② ボタンのメニューを出す

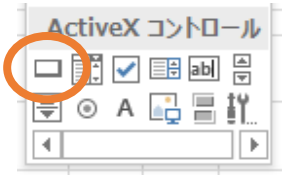
「**挿入**」からプルダウンメニューを**図 3-30** のように出します。

③ ActiveX コントロール

プルダウンメニューには「**フォームコントロール**」のボタン群と「**ActiveX コントロール**」のボタン群があります。プログラミングをしている場合「**ActiveX コントロール**」の中から選びます。

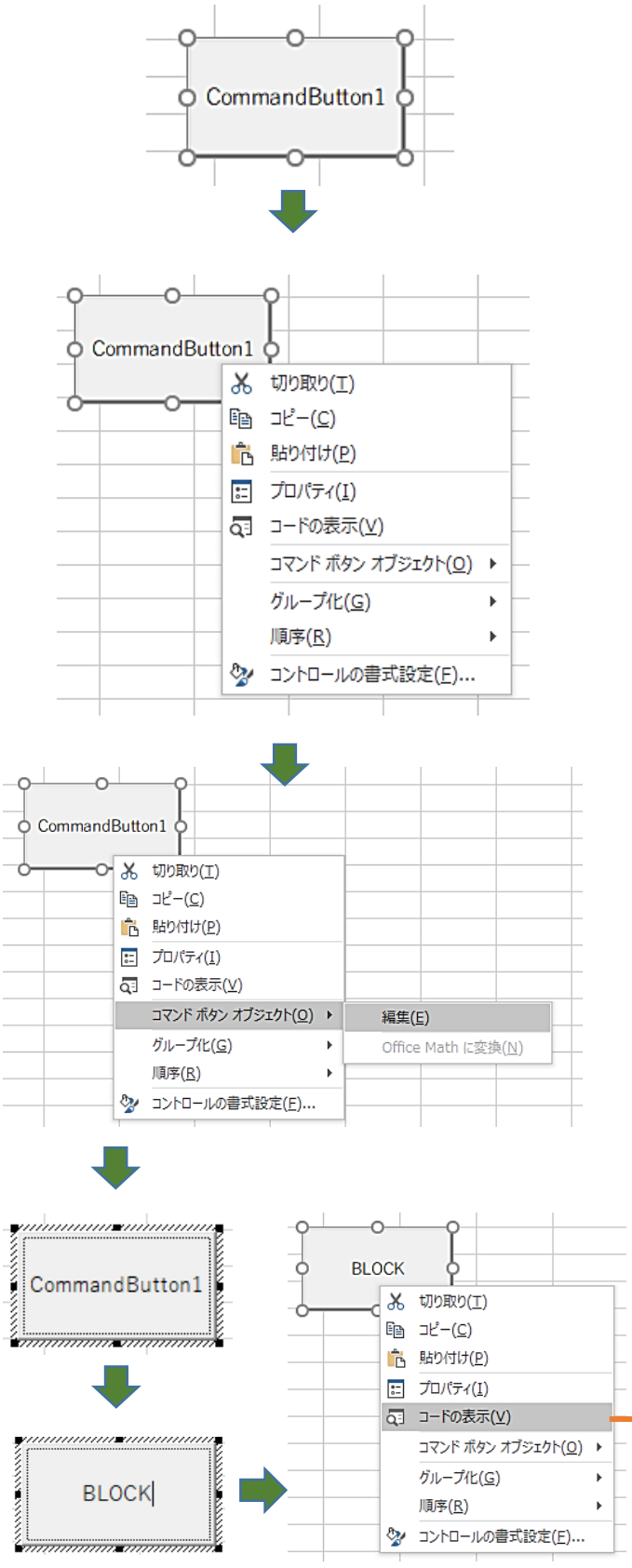
④ ボタンの形を選ぶ

「**ActiveX コントロール**」のボタン群の左上にある四角のボタンを選びましょう。



上のこの四角のボタンは、今回の講座で最終回まで使うことになる。この一連の設定手順はしっかり頭と体で覚えよう。

図 3-30



⑤ ボタンを広げる

マウスでボタンを広げて Excel シート上にマウスでボタンを広げます。CommandButton1 (コマンドボタン 1)というのは初めから書かれています。この名前を変えることにしましょう。図 3-31 の一番上

⑥ コマンドボタンオブジェクト

ボタンを右クリックしてプルダウンメニューから『**コマンドボタンオブジェクト**』から「**編集**」を選びます。

⑦ ボタンに BLOCK と書く

コマンドボタン 1 という名称を BackSpace キーで消して、**BLOCK** と書き直します。

⑧ ボタンにコードを書く

次に同じくプルダウンメニューから「**コードの表示**」を選ぶと、今まで書いていたプログラムコードのウインドウが出てきます。よく見ると一番下にコマンドボタン 1 のコードを書く場所が出てきています。

図 3-31

```
Private Sub CommandButton1_Click()
  For k = 1 To 9
     Cells(9+ k, 2) = k
     Cells(9+ k, 3) = 8
     Cells(18 + k, 2) = k
     Cells(18 + k, 3) = 9
  Next k
End Sub
```

日本語訳

k=1 から 9 まで繰り返して次のセルの値を替えなさい。最初に k=1 として

（Excel のシートのセルの）

セル（10,2）に 1 を入れなさい。

セル（10,3）に 8 を入れなさい。

セル（19,2）に 1 を入れなさい。

セル（19,3）に 9 を入れないさい。

次に k=2 として

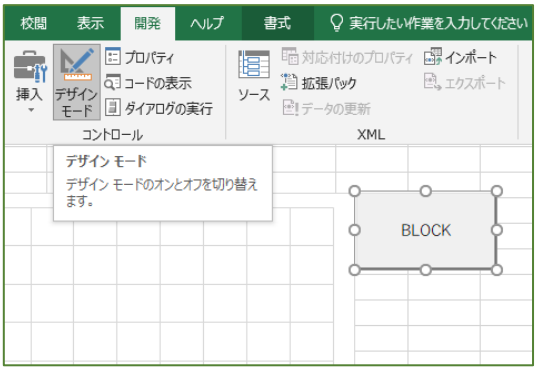
セル(11,2) に 2 を入れなさい。

セル(11,3) に 8 を入れなさい。

・・・

という具合に k=9 まで繰り返す。

するとブロックの座標のセルが初めの値に書き換えられ、ブロック 18 個が定位置に並ぶことになる。



⑨ デザインモードをオフにする

デザインモードをクリックするとオフになって濃かった色が消えます。これで **BLOCK** ボタンが機能するようになります。ボタンを押して下図 3-32 のようにパッとブロックが並んだらボタン設定成功！

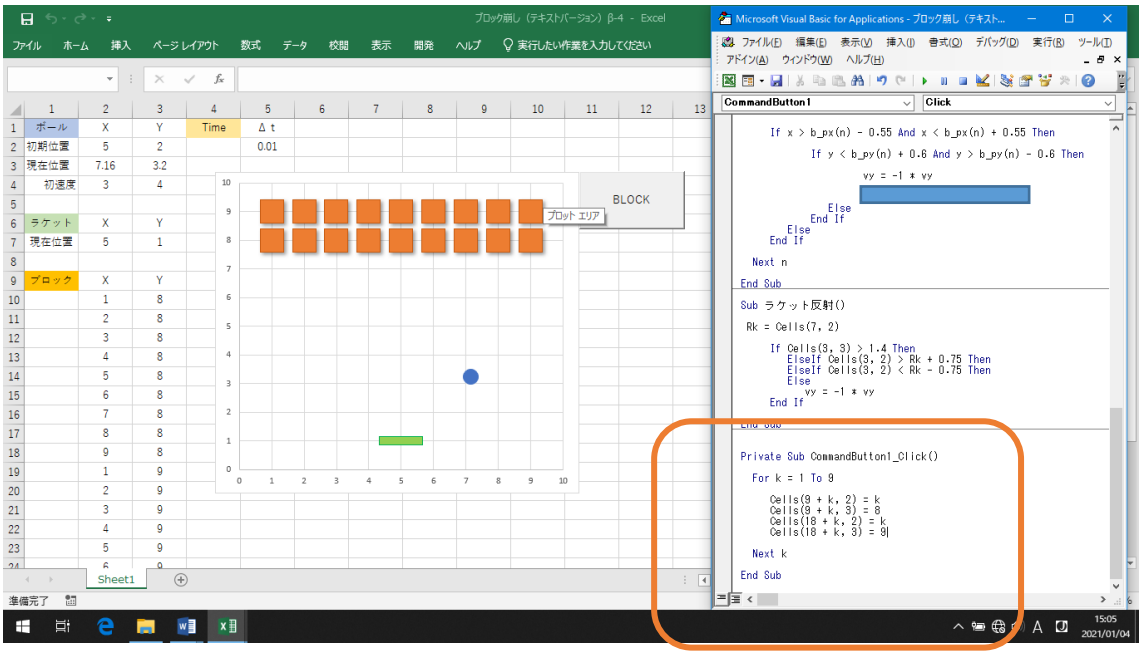


図 3-32

STEP 10　スタートボタンの作成

まだラケットが左右に動かないので早く動かすようにしたいだろうと思います。もう少し待ってください。その前に、このゲームのスタートとストップのボタンを一つのボタンで作ってみたいと思います。つまり、START を押すとゲームが始まり、その時ボタンの表示は START から STOP に代わるようにしておきます。そして、今度は STOP を押すとゲームが終わるようにするのです。このように一つのボタンで 2 つのことができると、いろいろ応用が利きそうです。では、早速作ってみましょう。

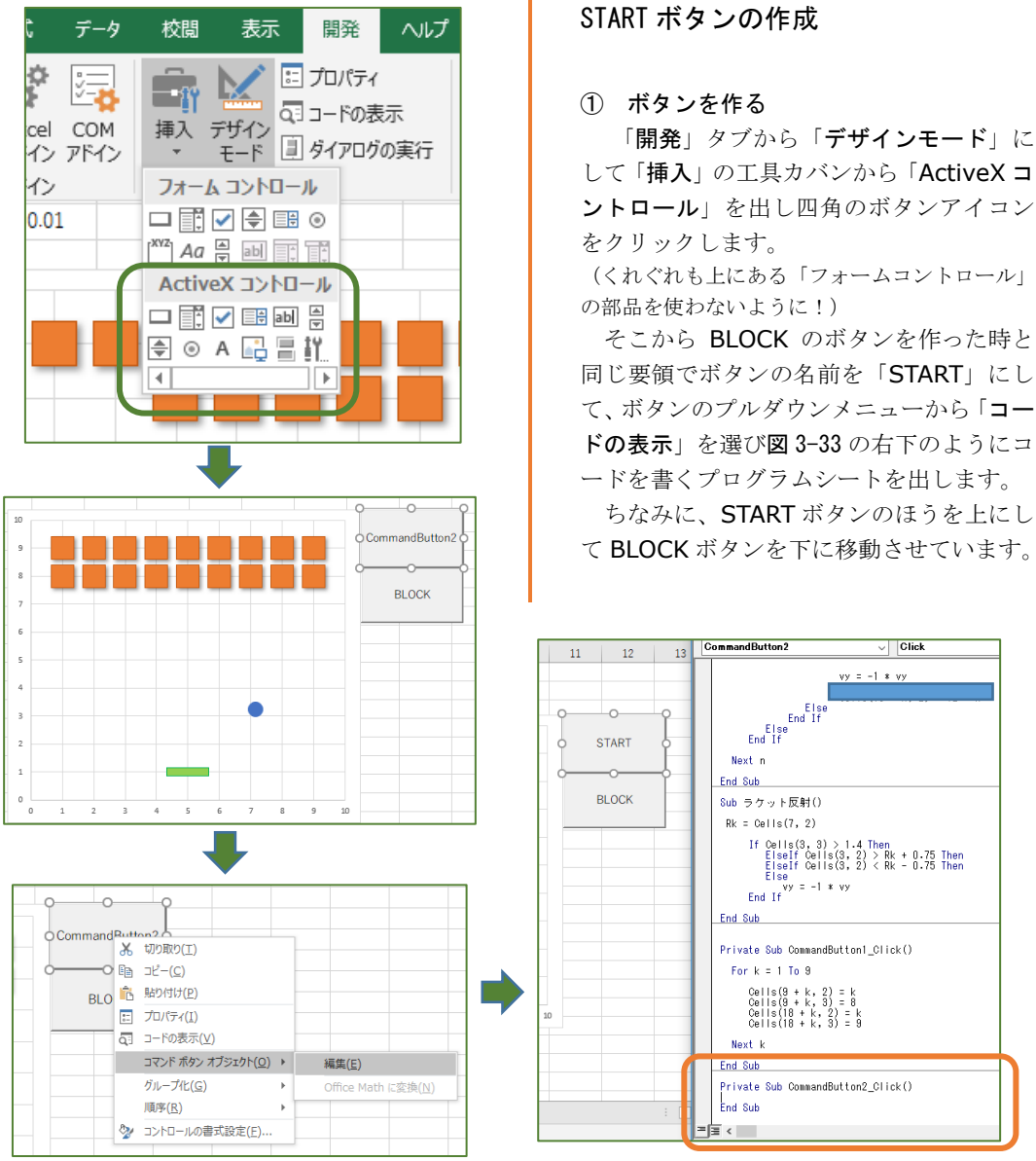


図 3-33

START ボタンの作成

① ボタンを作る

「**開発**」タブから「**デザインモード**」にして「**挿入**」の工具カバンから「**ActiveX コントロール**」を出し四角のボタンアイコンをクリックします。

（くれぐれも上にある「フォームコントロール」の部品を使わないように！）

そこから BLOCK のボタンを作った時と同じ要領でボタンの名前を「START」にして、ボタンのプルダウンメニューから「**コードの表示**」を選び**図 3-33** の右下のようにコードを書くプログラムシートを出します。

ちなみに、START ボタンのほうを上にして BLOCK ボタンを下に移動させています。

```
Private Sub CommandButton2_Click()

  If bContinue = True Then
        bContinue = False
        CommandButton2.Caption = "START"
    Else
        Call Ball
  End If
End Sub
```

日本語訳
このプログラムは Ball というメインプログラムに付属する（プライベートな）**命令ボタン２クリック**です。
If(もし) bContinue が「**真（トゥルー)**」なら
Then(そうであれば)
bContinue を「**偽（フォールス)**」として
命令ボタン２クリックの名称を「START」とせよ。
Else(それ以外なら)
Ball というプログラムを実行しろ。
End IF(以上)
EndSub(おわり)

このプログラムははっきり言って「意味不明」です。bContinue って何？真・偽ってなに？
分からないのは当然です。実は、この bContinue を「スイッチ」とみます。真と偽はこの「bContinue」スイッチの ON と OFF と考えれば読み解けます。
そして、次ページの全体のプログラムの説明に、この bContinue（スイッチ）が現れることで全体の意味が明らかになります。

まあ、このブロック崩しで一番わかりにくいところかもしれませぬ。頑張ってついてこいっ！次ページの翻訳を読んでみてください。

図 3-34

②　コードを書く

図 3-34 がボタン２「START ボタン」のコードです。ここで今まで出てきていない変数を導入しましょう。Boolean（ブーリアン）といわれるものです。

実数型(Single,Double)、整数型(Integer)のほかに変数のとる型があるのかと思うでしょうが、プログラムに特有の変数の型があります。

それは、0 と 1 しかとらない変数です。それを Boolean（ブーリアン）と呼んでいます。変な名前ですね。これは 1800 年代のイギリスの数学者ジョージ・ブールが作った 1 と 0 からなる代数をブール代数（boolean algebra）と呼んでいることからきています。ブールは論理（真か偽か）というものと数学の代数を結びつけた最初の人です。1864 年に肺炎で 49 歳の若さで亡くなりますが、彼の業績はそれから 80 年後に生まれたコンピューター科学で蘇（よみがえ）ります。オンかオフかつまり 1 か 0 かで回路演算をする方法が編み出され、コンピューターが作られていくのです。今作っているプログラムも、コンピューターの中では最終的に 1 か 0 の数値（**二進数**）に変換されて計算されます。大学の電子工学科とかに行くと、このブール代数に苦しめられることになるのですが…。

さて、図 3-34 にはそのブール数である Boolean 型になっている変数があります。それが bContinue です。「継続(continue) ボタン b」という意味でつけたもので長ったらしくて書きにくいと思いますが我慢してください。bContinue, START, BALL これだけは大文字小文字に気を付けて書けば、ほかの文字は小文字だけで変換してくれます。もちろん半角空白は必要です。

まずは、図 3-34 の下の日本語訳を読み、コードをエディターシートに書いていきましょう。

コードの追加と日本語訳

```
Dim vx, vy, dt, x, y, Rk As Single
Dim k, n As Integer
Dim b_px(100), b_py(100), b_vx(100), b_vy(100) As Single
Dim bContinue As Boolean

Sub Ball()

    （省　略）
 y = Cells(2, 3).Value

 bContinue = True
    CommandButton2.Caption = "STOP"

 Do
    （省　略）

 Loop While bContinue = True

End Sub

Sub ボールの壁反射()
    （省　略）
End Sub

Sub ブロック崩し()
    （省　略）
End Sub

Sub ラケット反射()
    （省　略）
End Sub

Private Sub CommandButton1_Click()
    （省　略）
End Sub

Private Sub CommandButton2_Click()

  If bContinue = True Then
        bContinue = False
        CommandButton2.Caption = "START"
    Else
        Call Ball
  End If
End Sub
```

a）宣言文の追加

b）コードの追加 1

まずこの Ball プログラムが走り出したとしましょう。
このとき bContinue（スイッチ）は真（True）つまり ON となってボタンの名前(caption)は「STOP」になります。
つまりスイッチが ON になって Ball プログラムの Do Loop が回り始めます。

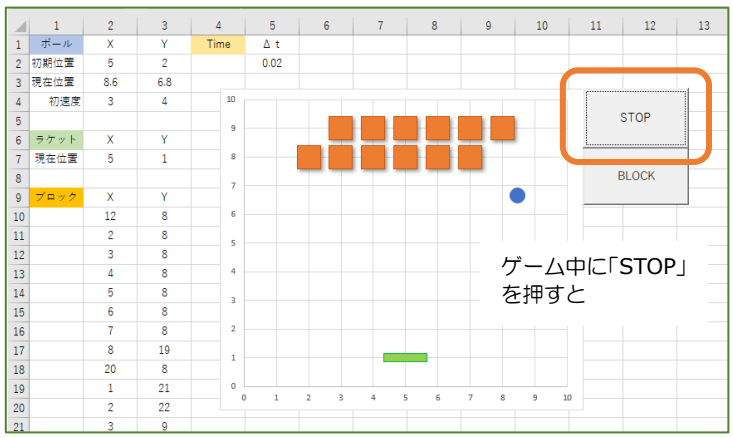
c）コードの追加 2

この bContinue（スイッチ）が(True) ON の間 Do Loop は回り続けます。逆に言えば、スイッチが OFF になると Do Loop が終わります。つまりゲームストップです。

d）プログラムの追加

「STOP」ボタンを押すと
もし bContinue（スイッチ）が真 ON のときは bContinue（スイッチ）を偽 OFF にしてボタンの名称は「START」になります。
すると、Loop からは外れて止まってしまうことになります。

再びゲームを再開するときはどうでしょう。
この時ボタンは「START」つまり bContinue は偽 OFF です。
この「START」ボタンを押すともし bContinue が真 ON のときはと聞かれても違いますので、Else(真以外のとき)に飛んで Ball プログラムを始めることになります。



③　デバッグと実行

コードを書いたら F8 でデバッグして初めてのボタンで実行してみましょう。うまく「STOP」と「START」のキャプション（名称）が変わりますか（**図 3-35**）。

これで完成に近づきました。ブロックもすぐに積み上げられますし、すぐにスタートできます。後はいよいよ「**ラケットを動かす**」だけです。

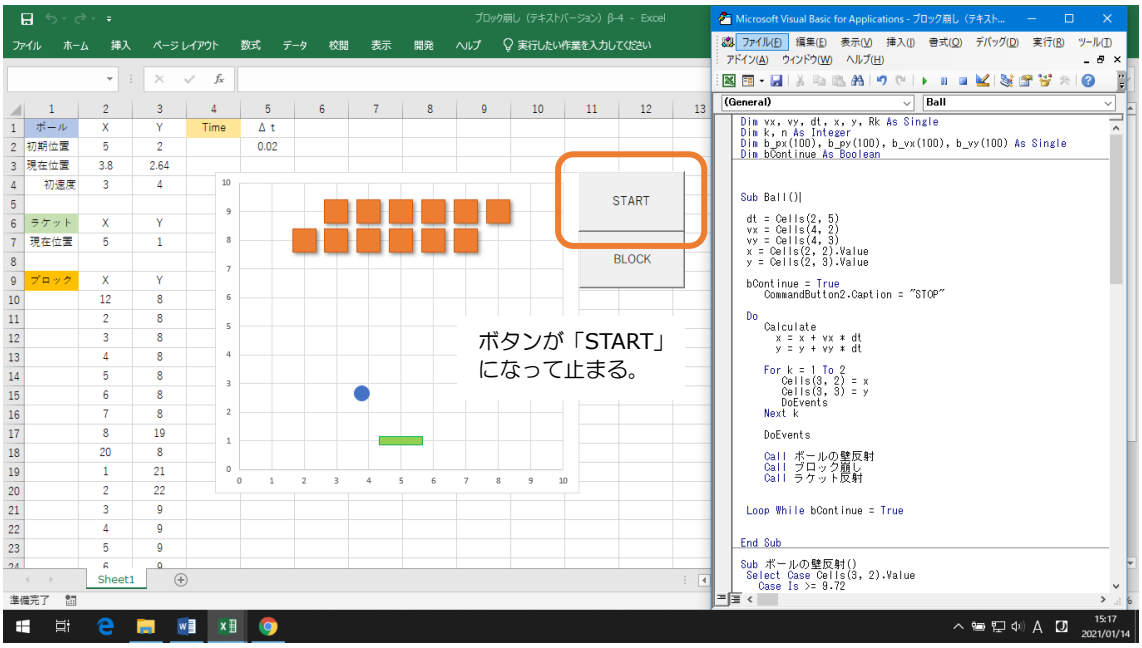


図 3-35

構造化プログラミング

改めてこのプログラムを見ると、まだラケットを動かすプログラムが完成していませんが、全体として Ball プログラムがメインになっていて、そこに壁とラケット、そしてブロックの反射条件を書いたプログラムが付属している構造になっていますね。

このように全体の構造が分かるように書いていくのもプログラムの大事なことです。このような概念を「**構造化プログラミング**」といいます。デバッグがしやすくなり生産コストパフォーマンスも上がります。また数学的な目で見ると、プログラム一つ一つは、入力があって出力があるという関数（ファンクション）です。このため「反射」という概念をもっと抽象化して、ラケットでもブロックでも壁でもその関数を通せば反射するというようにコード自体が発展していく可能性もありますね。このようなプログラミングの方向を**関数型プログラミング**といいます。さらにはデータと関数をまとめてオブジェクトとしてそれらのオブジェクト部品を関連させてプログラミングするオブジェクト指向型プログラミングという方向もあります。この代表が Python（パイソン）です。

STEP 11　ラケットを動かす

さて、あとはラケットを動かすだけになりました。この動かす方法については 2 種類紹介したいと思います。まず Mac を使っている人も Windows を使っている人も使えるボタンを使って動かす方法です。もう一つは、キーボードの ← → キーを使ってラケットを動かす方法です。これはラケットとの一体感を生み出すのですが、残念ながら Windows マシンにしか使えません。**ボタン型**と**キーボード型**という呼び方で分けたいと思います。これらの両方とも実装しますので、テキストの順に作業をしていきましょう。

1．ボタン型

それでは、**「開発」**から**「デザインモード」**を選び、**「挿入」**の工具箱から ActiveX コントロールのメニューを出して、四角のスイッチを選びます。（しつこいようですがくれぐれも**「フォームコントロール」**から選ばないこと。**マクロ設定**に行ってしまいます。）

四角スイッチを広げて、アクティブにしたまま（周りが泡が出ている状態）右クリックして**『コマンドボタンオブジェクト』**の**「編集」**を選び、このスイッチの名前を「LEFT」としておきます。

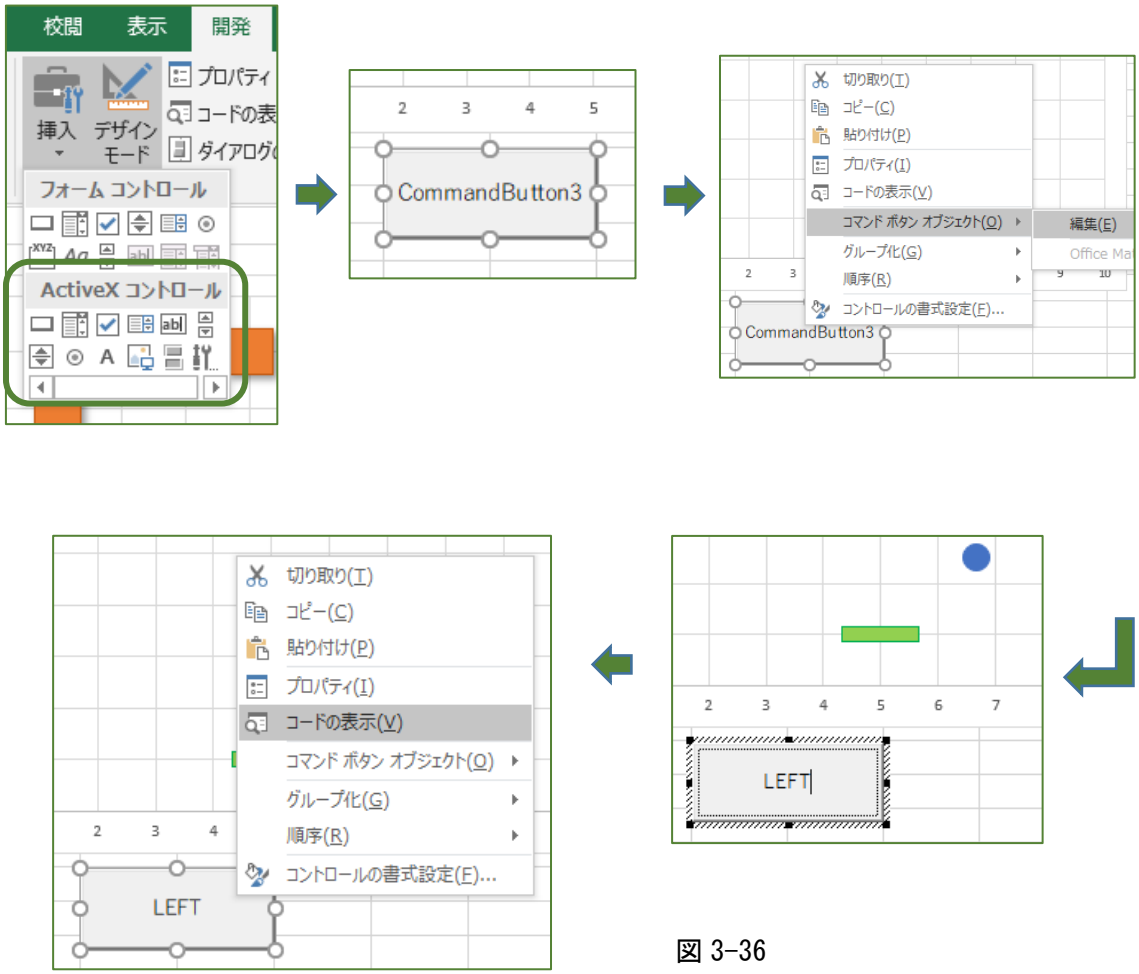
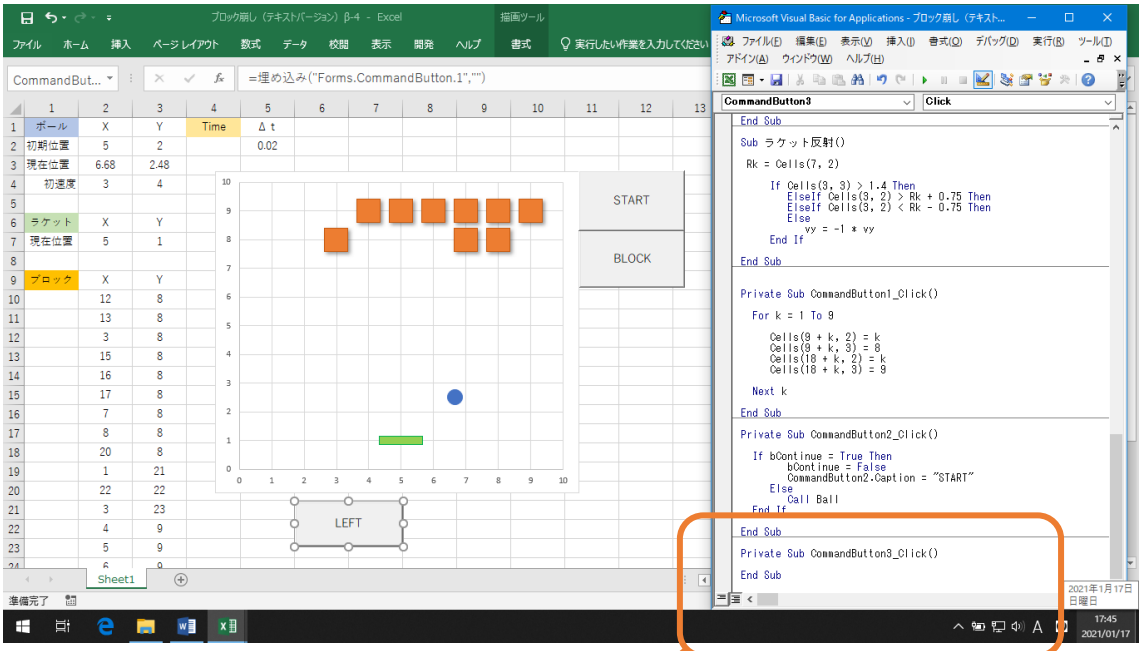


図 3-36




```
Private Sub CommandButton3_Click()
    Rk = Cells(7, 2).Value
    Rk = Rk - 1.4
    Cells(7, 2).Value = Rk
    DoEvents
End Sub
```

図 3-37

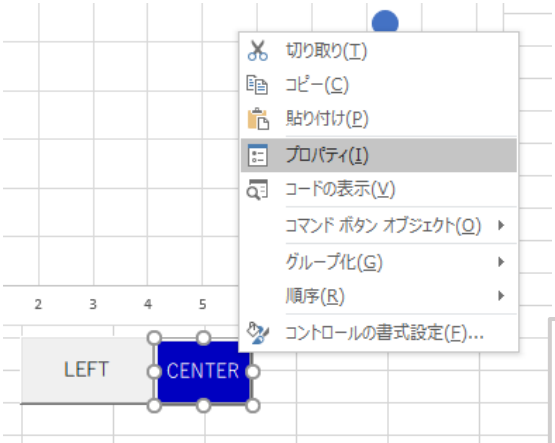


図 3-38

LEFT スイッチのコード

LEFT スイッチをクリックすると左に 1.4 目盛ほどラケットが移動してくれるようにします。**図 3-37** のコードはもう難しくないですね。**セル(7,2)**の値を Rk というラケットの X 座標を表す変数にいれて、次にその Rk の値から 1.4 を引いた値を、新しい Rk の値とします。それを、再び**セル(7,2)**に入れてやって、グラフの作図をさせます。(DoEvents)

CENTER スイッチのコード

次に LEFT スイッチの右側のちょうど真ん中に当たる 5 目盛の所に CENTER スイッチを作ります。これは単純なコードです。**図 3-38 右**にあるように、このスイッチを押すとラケットが中央に来るようにするだけです。大きく左右にラケットが振られた時にセンターに一瞬にして戻れるスイッチです。

```
Private Sub CommandButton4_Click()
    Cells(7, 2).Value = 5
End Sub
```

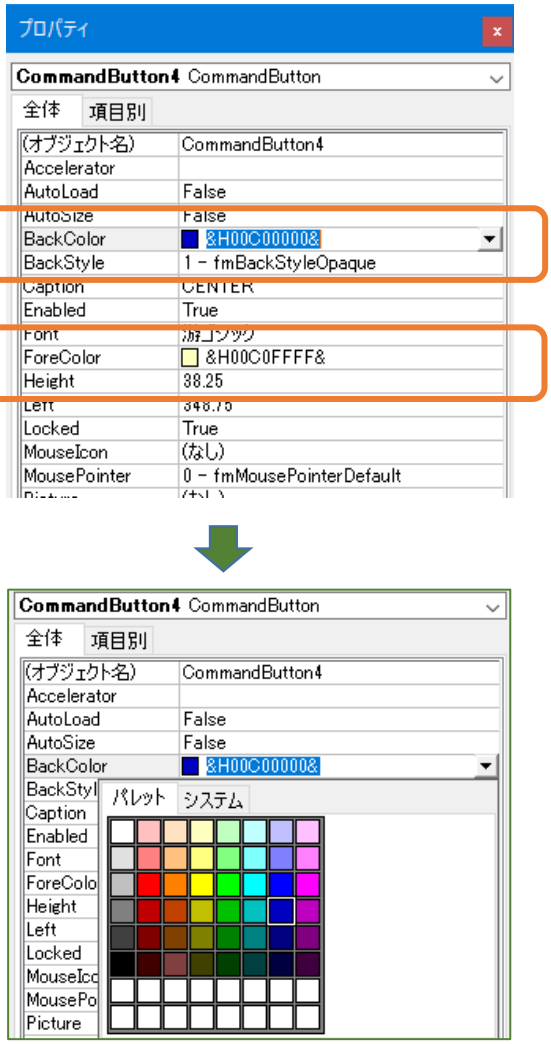


図 39

CENTER スイッチのデザイン

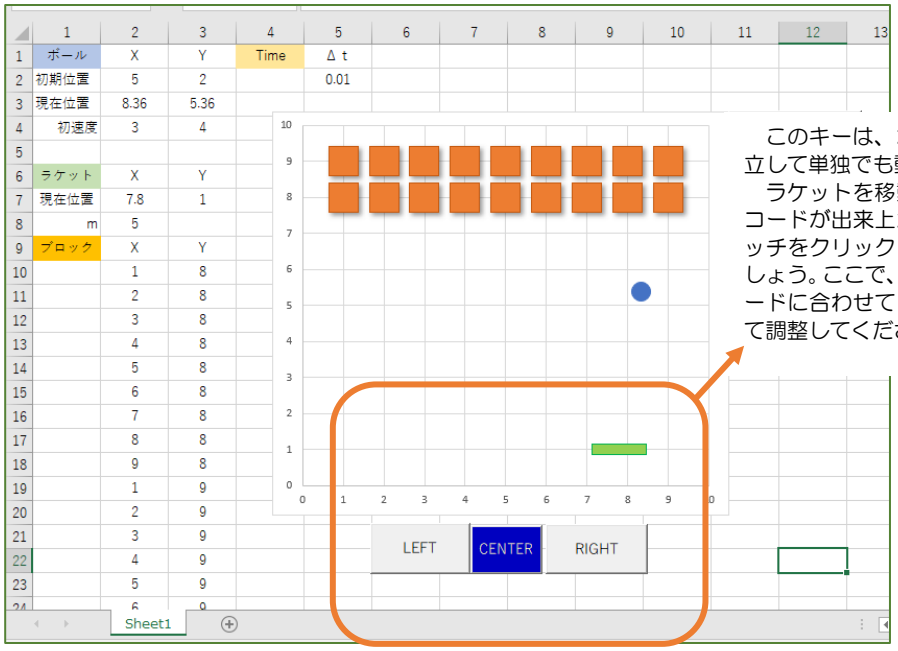
図 3-38 を見て気付いたと思いますが、CENTER スイッチの色が違っていますね。スイッチのデザインの詳細（プロパティ）は図 3-38 のように右クリックして「プロパティ」をクリックすると図 3-39 上が出てきます。何も知らないと細かくて「おおっ」とのけぞりそうですが、よく見るとそんなに意味不明ではありません。そこに BackColer（背景色）とあるところをクリックして詳細を見てみると、色のパレットが出てきます。ここで好きな色を選ぶだけです。図では青を選んでいます。同様にして ForeColer も選んでいます。スイッチを違う色にしてどれがどのスイッチだったかを判断しやすいようにするためです。

RIGHT スイッチをつくる

図 3-40 を見てください。RIGHT スイッチも作られていますね。これは、自分で作れると思います。このスイッチを押すと右に 1.4 目盛ほどラケットが移動するスイッチを作ってください。

問題 3-3

ラケットを右に移動させるスイッチを作りなさい。



このキーは、ボールの動きとは独立して単独でも動きます。

ラケットを移動させるスイッチのコードが出来上がったら、早速スイッチをクリックして移動させてみましょう。ここで、自分のマシンのスピードに合わせてコードの数値を変えて調整してください。

図 3-40

１．キーボード型

キーボード型のラケットコントローラーは、**タイムラグ**（時間的遅れ）がなく、押し続けると等速で移動しますので、ゲームに適しています。ただ、残念ながら Mac には使えません。現在、世界のパソコンの基本的な**オペレーションシステム**（OS）は、大きく二種類に分かれます。1970 年代にマイクロソフト社のビルゲイツたちとアップル社のスティーブジョブズたちが競争しながら開発していったパソコンの基本 OS は、今 Windows と MacOS に進化しました。実は、初期のころに日本からも世界と競える OS が提案され、これは現在ではパソコン以外のものに数多く搭載されています。

パソコンとスマホやタブレットとの違いは、パソコンのほうがより研究や創作するための道具としての色彩が強いということです。実際、プログラミングなどはパソコンを使います。また、画像や動画の創作、論文や小説などの執筆などはパソコンのほうが向いています。ま、スマホで論文を書く人もいるのかもしれませんが…？

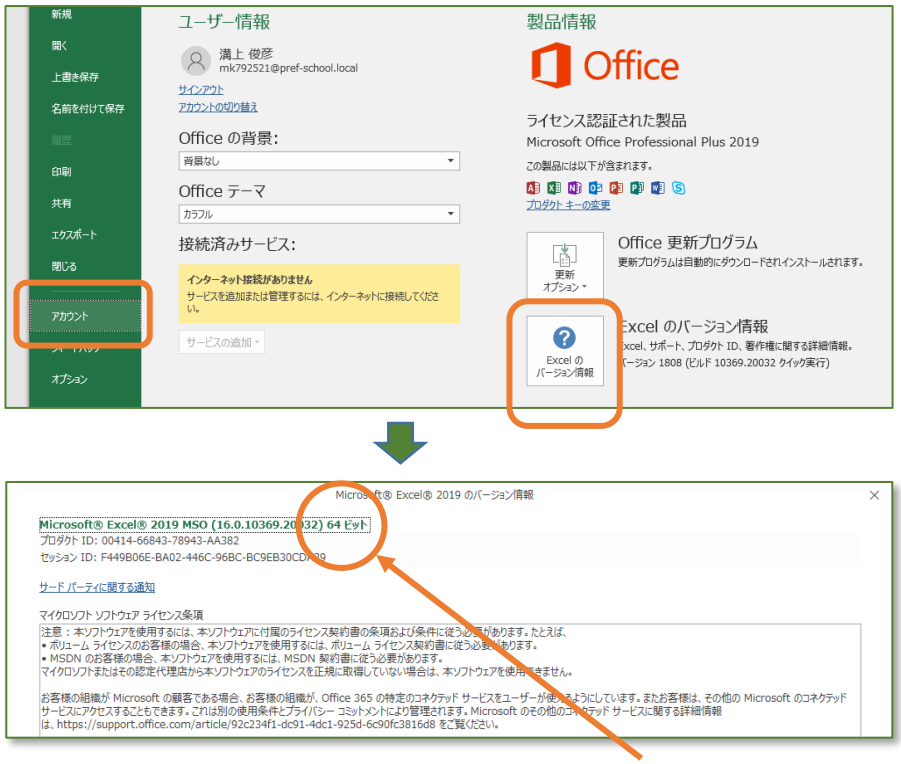
それでは Windows マシン（Mac の**リンゴ**がついてないやつ）の人はキーボード型のラケットコントローラーを作ってみましょう。これを実現するには、コードを少し付け加えるだけです。まず準備をしましょう。

準備 1

まずキーボードを使って直接コントロールしますので、パソコンの心臓部にかなり近づきます。そのために、Excel への「**挨拶**」が必要です。

まず、君の使っている Excel は 64bit 版か 32bit 版かを調べます。**図 3–41** を上から読んで確認してください。bit（**ビット**と読みます。1 と 0 の数字で 64 桁を作り 2^{64} 通りの情報を扱えるのが **64bit 版**で

Excel の「**ファイル**」タブから「**アカウント**」をクリックし下図のダイアログを出す。
この中の「**Excel のバージョン情報**」の ⓘ をクリック。



ここにビット数が書いてある。図の場合は 64bit

図 3–41

１．キーボード型（続き）

準備２

キーボードを使うというのは、ソフト的なボタンよりもよりリアルな反応が期待できます。キーボードには、プログラミング用に一つのキーに一つの「**キーコード**」が割り当てられています。（キー入力判定用 API といいます。）下の表（図 3-42）を参考にしてください。

今回はこの中のキー ← (37) と → (39) を使います。

この「**キーコード**」を使うには、プログラムの前に使うことを宣言するコードを書く必要があります。意味は不明で魔女のおまじないのようなコードですが、以下にそのコードと入力の仕方を書きます。

入力の仕方と魔女のコード

プログラムの一番最初に書きます。小文字で書いても次の文字以外は大文字小文字を自動で変換してくれます。変換しているかどうかで魔女のコードの正誤のチェックをしましょう。

例外　GetAsyncKeyState　User32.

64bit の場合

```
Private Declare PtrSafe Function GetAsyncKeyState Lib "User32.dll"
 (ByVal vkey As Long) As LongPtr
```

32bit の場合

```
Private Declare Function GetAsyncKeyState Lib "User32.dll"
 (ByVal vkey As Long)
```

キー	コード	キー	コード	キー	コード	キー	コード
BackSpace	8	Insert	45	F	70	W	87
Tab	9	Delete	46	G	71	X	88
Enter	13	0	48	H	72	Y	89
Shift	16	1	49	I	73	Z	90
Ctrl	17	2	50	J	74	*	106
Alt	18	3	51	K	75	+	107
Pause	19	4	52	L	76	-	109
Esc	27	5	53	M	77	.	110
Space	31	6	54	N	78	/	111
PageUp	33	7	55	O	79	F1	112
PageDown	34	8	56	P	80	F2	113
End	35	9	57	Q	81	F3	114
Home	36	A	65	R	82	F4	115
←	37	B	66	S	83	F5	116
↑	38	C	67	T	84	F6	117
→	39	D	68	U	85	F7	118
↓	40	E	69	V	86	F8	119

図 3-42

```
Private Declare PtrSafe Function GetAsyncKeyState Lib "User32.dll" (ByVal vkey As Long)
As LongPtr

Dim vx, vy, dt, x, y, Rk As Single
Dim k, n, m As Integer
Dim b_px(100), b_py(100), b_vx(100), b_vy(100) As Single
Dim bContinue As Boolean

Sub Ball()
        dt = Cells(2, 5)
        vx = Cells(4, 2)
        vy = Cells(4, 3)
        x = Cells(2, 2).Value
        y = Cells(2, 3).Value

        Rk = Cells(7, 2)

        bContinue = True
      CommandButton2.Caption = "STOP"
   Do
        Calculate
        x = x + vx * dt
        y = y + vy * dt
      For k = 1 To 2
          Cells(3, 2) = x
          Cells(3, 3) = y
          'DoEvents
      Next k
      'DoEvents
      Call ボールの壁反射
      Call ブロック崩し
      Call ラケット反射
      DoEvents
   Loop While bContinue = True
End Sub

Sub ボールの壁反射()（省略）
Sub ブロック崩し()　（省略）

Sub ラケット反射()
  If GetAsyncKeyState(37) > 0 Then
            Rk = Rk - 0.1
  End If
  If GetAsyncKeyState(39) > 0 Then
            Rk = Rk + 0.1
  End If

  Cells(7, 2) = Rk

   If Cells(3, 3) > 1.4 Then
      ElseIf Cells(3, 2) > Rk + 0.75 Then
      ElseIf Cells(3, 2) < Rk - 0.75 Then
      Else
          vy = -1 * vy
   End If
End Sub
Private Sub CommandButton1_Click() ～ Private Sub CommandButton5_Click()　（省略）
```

a）キーコード宣言文

これは 64bit の宣言文。
改行せずに最後まで書きます。

b）コードの追加１

ラケットの x 座標であるセル(7,2)の値を初期値として Rk というラケット変数に入れます。

c）コードの追加２

GetAsyncKeyState(37) は左向きの ← を意味しています。もしこのキーが押されたら（>0）
Rk を 0.1 減らして新しい Rk にしなさい。
同様に
GetAsyncKeyState(39) は右向きの → を意味しています。もしこのキーが押されたら（>0）
Rk を 0.1 増やして新しい Rk にします。

d）コードの書き直し

前回まで
Rk = Cells(7,2)
だったコードを左右を入れ替え
Cells(7,2) = Rk
とします。
Rk の値がキーボード入力で変わっていくので、その値を順次セル(7,2)に一旦納めないと、反射の判定ができないからです。

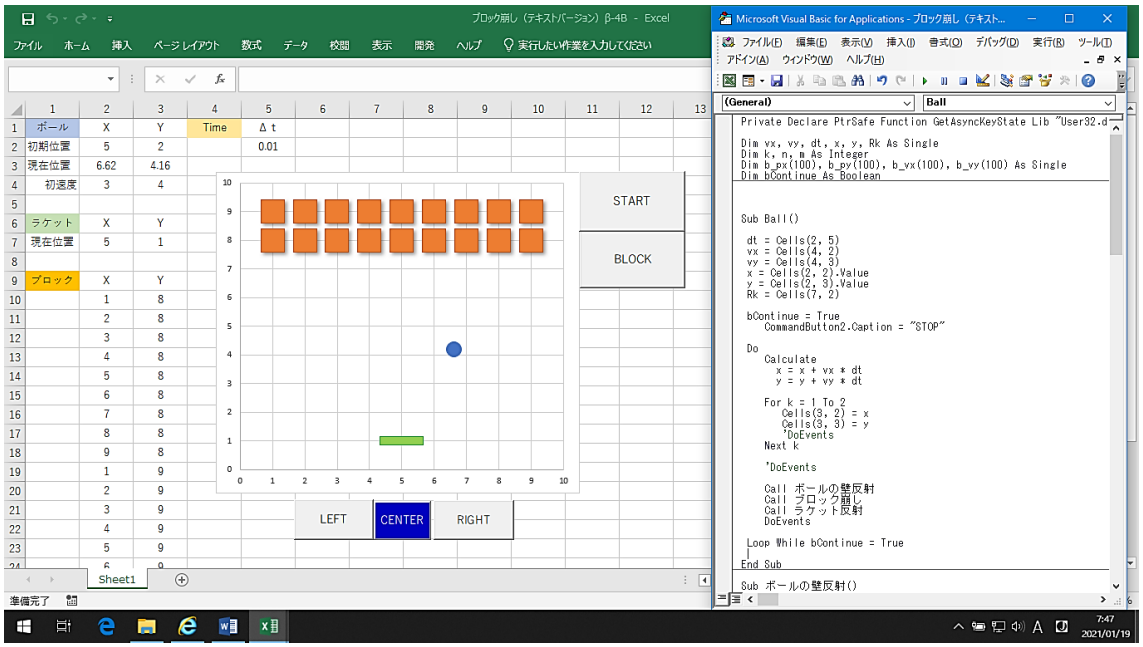


図 3-43

完成した「ブロック崩し」
キーボードからもボタンからもラケットをコントロールできる。ボールのスピードは、初速度や Δt を使って変えられるが、自然に動くようにするにはプログラムによる時間コントロールが必要。

課題Ⅴ　オリジナルゲームの開発

課題　「ブロック崩し」を改良して自分だけのオリジナル作品を作れ。

「ブロック崩し」が動き出したとき、今まで赤の他人だったパソコンが「自分の道具」に近づいたように思った人も多いだろう。中には、道具というより知性を持った何か、というような不思議な感覚を持った人もいるかもしれない。

「ブロック崩し」をしているといろいろなアイデアが浮かぶ。ブロックの位置をランダム（無作為）にボタン一つで配置できないのかとか、ラケットをブロックを挟んで 2 つにして二人でやるブロック崩しテニスゲームにできないのか、とか・・・。

もともと、大ブームを起こしたブロック崩しゲームのだいご味は、実はブロックの配置にあった。はじめは単純にブロックが並んだものが多かったが、そのうちブロック数が増え、配置が換わっていく。例えば、上の段と下の段のブロックの距離を少し広げると、ボールが下の段を壊してその間に入っていき、上下にジグザグに次々と一気にブロックを壊していくというのもあった。ゲーマーは、このキーになるブロックだけを狙って一点突破を目指す。作者はなかなかこれができないように工夫する。突破できたら一網打尽（いちもうだじん）の面白さを伝えることができる。

君も、自分なりのプログラミングの工夫をこの「ブロック崩し」に加えてくれ。もちろん、こんな発想があったのか！やられた！という作品はみんなの称賛を浴びることになるだろう。

STEP 12　VBA 基本構文の理解

問題 3-4　次の「BASIC 基本構文」１から 7 までのコードを実行するとセル(3,3) は
どんな値になるか考えて答えなさい。正解はコードを実際に Excel に書いて仲間と共に確認しなさい。

ヒント　Then（そうならば）　Else（それ以外）　ElseIf（それ以外でもし）
Value（値）

BASIC 基本構文１

```
X = 0
  For n=1 To 5
    X = X + 5
  Next n
Cells(3,3) = X
```

BASIC 基本構文２

```
X = 0
  For n=1 To 5
    X = X + 5 * n
  Next n
Cells(3,3) = X
```

BASIC 基本構文３

```
X = 0
  Do
    X = X + 30
  Loop While X < 100
Cells(3,3) = X
```

BASIC 基本構文４

```
X=3
IF X + 5 > 10 then
  X = X + 5
End IF
Cells(3,3) = X
```

BASIC 基本構文５

```
X=3
  IF X + 5 > 10 then
  X = X + 5
Else
  X = (X+5) ^ 2
End IF
Cells(3,3) = X
```

BASIC 基本構文６

```
X= -3
IF X + 5 > 10 then
  X = X + 1
ElseIf  X+5 > 5 then
  X = X + 5
ElseIf  X+5 >0 then
  X = X + 10
Else
  X = X + 15
End IF
Cells(3,3) = X
```

BASIC 基本構文 7

```
X=3
Y = X ^ 2 + 2 * X + 3
Cells(3,4) = Y
Select Case Cells(3,4).Value
  Case Is >= 20
    Z = 1
  Case Is <= 10
    Z = 2
  Case Else
    Z = 3
End Select
 Cells(3,3) = Z
```

解答確認コード例

```
Dim X, Y, Z As Single
Dim n As Integer
Sub TEST()
  X = 3
  Y = X ^ 2 + 2 * X + 3
  Cells(3, 4) = Y
    Select Case Cells(3, 4).Value
      Case Is >= 20
          Z = 1
      Case Is <= 10
          Z = 2
      Case Else
          Z = 3
    End Select
      Cells(3, 3) = X
End Sub
```

最後に

AppleⅡは 1993 年までに世界中で 500 万台が売れたといわれます。

その頃、僕の通っていた大学の冷暖房完備の計算機センターの中では、何億円もする TOSBAC という大型コンピュータが、そのセンターの王として君臨していました。しかし、王様の姿を見たものは仲間の中には誰もいません。物理の研究でこの計算機センターを使っていた僕も、卒業までこの TOSBAC という王様の実際の姿を見ることはありませんでした。受付カウンターの美女とは話せてもカウンターの中には入れなかったのです。

プログラムコードを読み込ませるだけでも、穿孔機（せんこうき）という機械でマークシートのような紙のカードに穴をあけ、それを束にした形で機械に読み取ってもらう必要がありました。そして最後にプリントアウトした結果を係りの人からもらう、という気の遠くなるような手間がかかっていました。しかも係の人からもらった時エラーが出ていて結果が出ていないこともしょっちゅうでした。

そんな時代に AppleⅡは登場したのです。それは、その 10 年ほど前、金持ちしか手にすることのできなかった高級車の時代に、僕らにも手の届く軽自動車ホンダ N360（当時 31 万円ぐらい なんと空冷エンジン）が発売された状況に近かったと思います。革命が起こったのです。自分の手で好きなようにコンピューターが扱える、それは夢のようでした。自分で拙いコードを組み合わせてプログラムを書き、自由に自分の研究のデータ処理ができるというのは、まさに王様の気分だったのです。

AppleⅡは今の君たちが見ると、笑ってしまうほどのスペックでした。高解像度グラフィックスといいつつ、ディスプレイは 280 ドット×192 ドット、しかも色は 6 色。現在のマシンは、1677 万色なんて普通。しかし、僕らは狂喜しました。それまでそんなグラフィック機能を持ったコンピューターを見たことさえなかったからです。あの計算機センターの王様でさえ、紙でしか返事をしてくれなかったのだから・・・。

それでも当時 30 万円以上した AppleⅡは学生には高く、僕は就職したら絶対買うぞと心に誓いながら、プログラムできる電卓（当時 2, 3 万円した）をバイトして買いました。簡単な実験のデータ整理はいつもそいつで行いました。友達とそれを使って「潜水艦ゲーム」をつくり、何度も彼から魚雷で撃沈させられもしました。もちろん、僕もお返しをしましたが…。

時を同じくして、若者の中から「天才プログラマー」と呼ばれる人たちが現れ、斬新なゲームが発表されだしました。喫茶店には「ブロック崩し」やあの「インベーダーゲーム」が、ディスプレイを埋め込んだガラステーブルの形で登場しました。静かだった喫茶店は、やたらうるさい電子音にあふれるようになり、僕は自然と喫茶店から足が遠のいていきました。

プログラムコードをちょっとでも書いた人ならわかるのですが、コードが書ける人でコンピュータゲームに夢中になる人は少なくとも僕の周りにはいませんでした。ゲームをするより創るほうがよっぽど面白いし、研究に役立つプログラムコードを考えるほうがもっとエキサイティングだったからです。仲間には卒業してコンピューターエンジニアになった連中もたくさんいました。

その後世界は大きく変わっていき、世界中のパソコン 1 台 1 台がネットワークでつながってしまうという、誰も予想しなかった進化を遂げます。しかも小中学生までもが、「スマホ」といわれるパソコンを持つようになり、プログラミングをする人たちはオタク扱いされるようになっていきます。閲覧と通信の道具になり果てたコンピュータを日本人が手にして騒いでいるころ、世界では人工知能の研究が進み、気が付くと日本は大きく後れを取っていることが明らかになってきました。

慌てた日本では、小学校へのプログラミング学習の導入が叫ばれるようになり、この「失われた 10 年」を取り戻そうとしています。

当時の AppleⅡは、大型コンピューター王国の独裁政治を倒した革命の最新兵器でした。でも、その革命が日本ではコンピューター文化の民主化に本当の意味ではつながらなかったのだと思います。また大きな力でプログラミング教育が始まろうとしています。子供たちが、この「機械と話せる言語」を嫌いにならないでほしいなと思います。

今ではいい年のじっちゃんになったこの僕は、AppleⅡが作られた時代とともに年を取ってきました。まさかこの年になって、プログラミングの基本を若者に教えるためのテキストを書くことになるとは想像もしていませんでした。

このプログラミングテキストが、君たちの未来に少しは役に立っていくことを願っています。君たちの中から、次の革命を起こす道具を発明してくれる天才が出くることを願ってこのブロック崩しの章を終わります。

それじゃ。

2021 年 1 月 14 日

きみろん Comp. 第 4 章

— モンテカルロ法 —

モンテカルロ法

モンテカルロ法というのは、コンピュータを使って確率的な方法で問題を解く方法です。考案したのは、コンピュータの父といわれるフォン･ノイマン(V.Neumann)　とウラム(S.M.Ulam) といわれています。現在では、コンピュータの発展とともに、問題を数学的モデルとして定式化できない、できても解くことのできない問題を、乱数を使って解く方法一般を指す言葉になっています。

いわばコンピュータの中でサイコロを振るこのモンテカルロ法を使うと、現実の世界に近いよりリアルな状況でシミュレーションすることができるようになります。

ここでは、まず「**コンピュータの中でサイコロを振る**」というのはどういうことなのかということを学ぶために、確率的に円周率πを求めるということをやってみましょう。