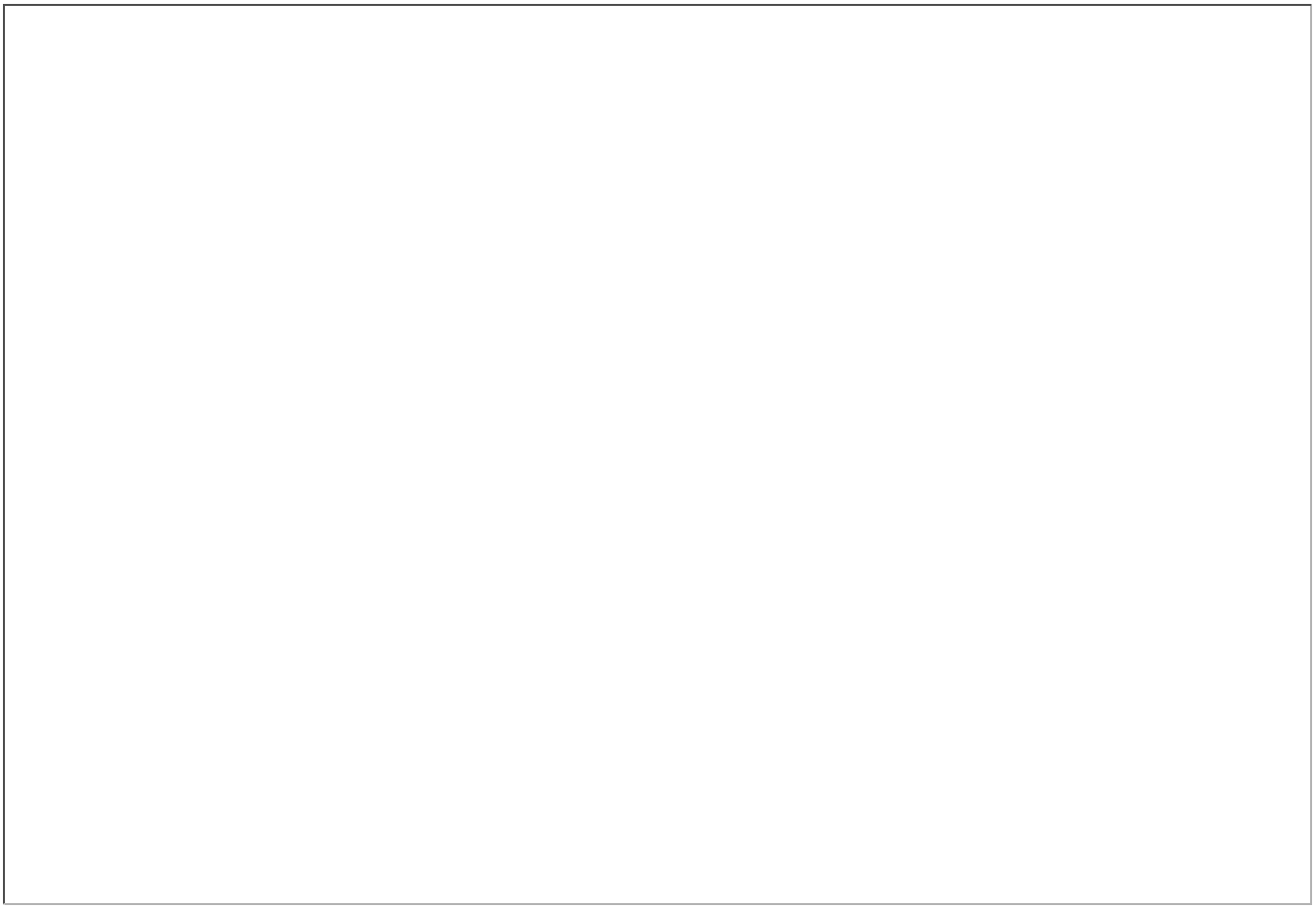
Learning 2D Game Programming: Getting Start Part 1 - Writing your first BlitzMax Program

Table of Contents

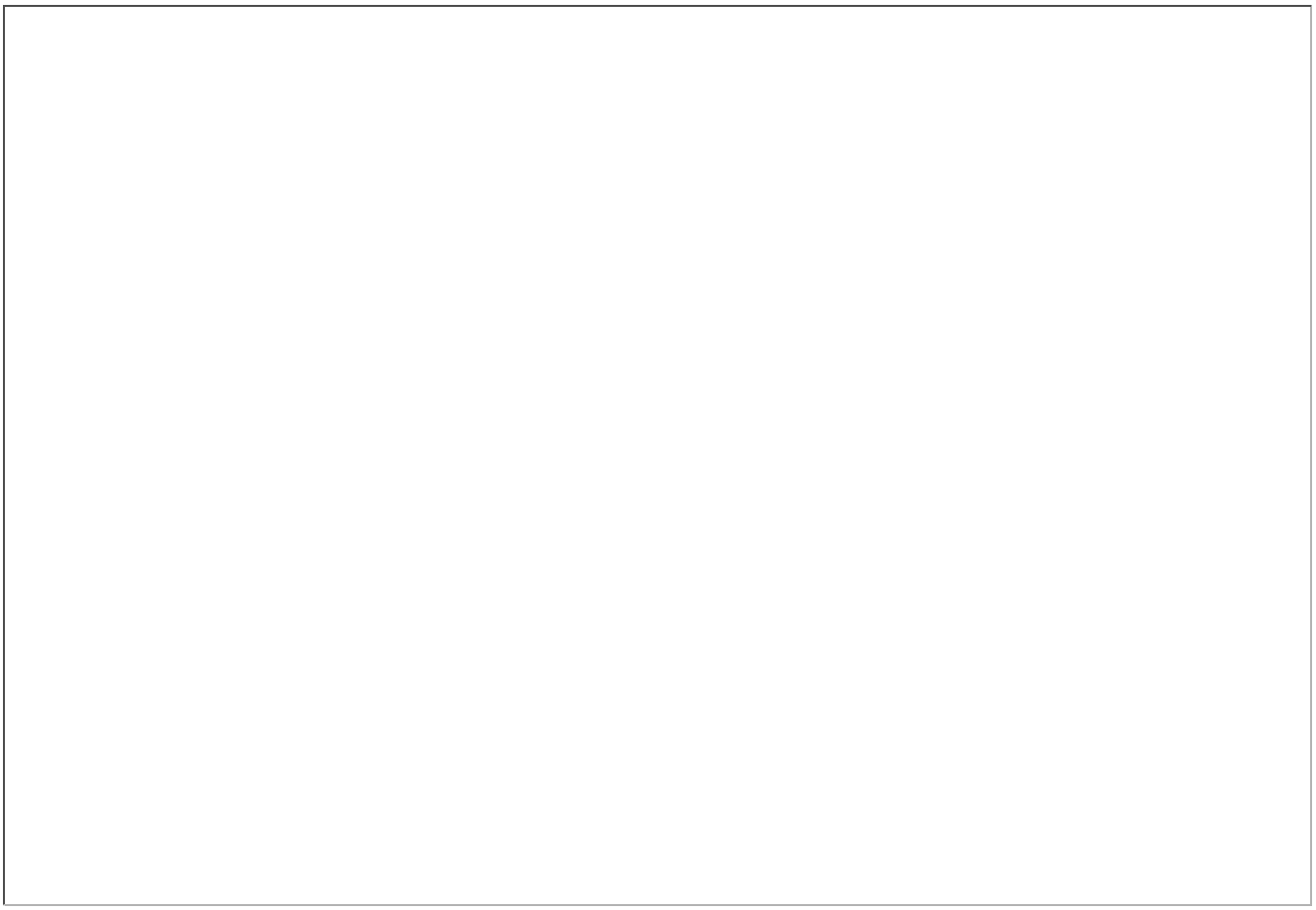
Introduction

So you have just downloaded BlitzMax and has successfully installed the program and fired up the MaxIDE. So what do you do next?

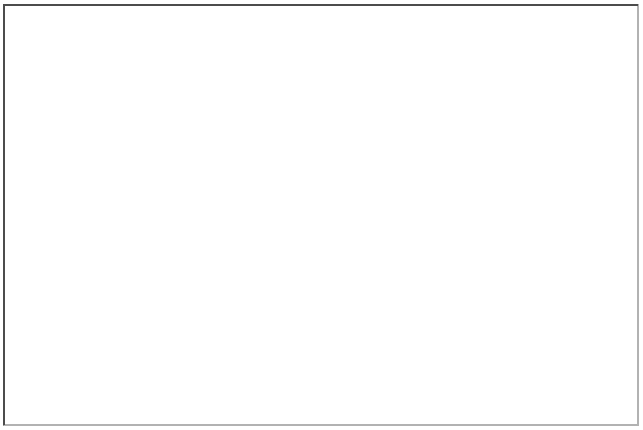
If you are staring at a screen that looks like this then we can start doing something with BlitzMax.



Before we go any further, let us familiarise ourselves with the BlitzMax Integrated Development Environment (MaxIDE). Click the button as pointed by the arrow below. This will create a new blank file to work on.

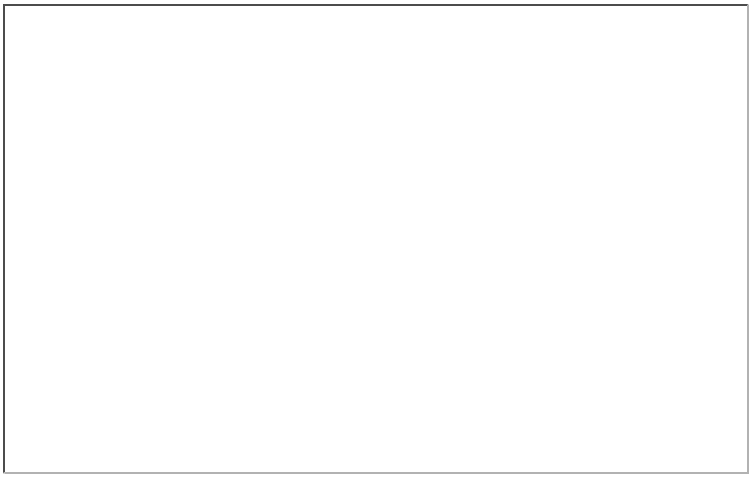


You should then get a blank space to work on as per the screen below (the colors and font may be different on your system). Note that there is now a new tab called **untitiled1.bmx**



Let us create the traditional "Hello World" program. Key in the text into the workarea and then clicked on the run button as shown by the arrow:-

The action now shifts to a new tab called **Output**. The short program you have just written (congratulations, by the way) is now being processed into an executable file called **untitled1.debug.exe**. The text "Hello World" is then printed onto the **Output** tab area.



All I wanted to do here was just to be sure you know how to create and run a BlitzMax program. We are now ready to proceed with the tutorial.

Our First BlitzMax Program

Our first BlitzMax program consists of the following 5 lines. Click on the New button and type these 5 lines into the MaxIDE

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

Before you compile and run the above program, note that to exit from the program and return to the Blitzmax editor, you will need to click the mouse button on the graphic screen. This is true for all the programs in this set of tutorials.

Now compile and run the program by clicking the "Build and Run" button as you did above. This is what we get:-

Let us try to understand what we have done. We actually have issued 5 instructions to BlitzMax.

The first instruction,

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

tells BlitzMax to create a graphic screen sized at 640 pixels wide and 480 pixels high.

The second instruction,

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

tells BlitzMax to draw at the screen coordinates 50,50 a rectangular box whose sides are 200 pixels wide and 100 pixels high. See diagram below.

The **DrawRect** instruction draws to a temporary location which was hidden from our view. The third instruction,

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

tells BlitzMax to display the hidden screen by **flip**ping it into view.

The fourth instruction,

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

simply tells BX to wait for us to click the mouse button before moving to the next instruction.

And finally the last instruction simply end the program.

```
Graphics 640,480,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

So there we have it our first BlitzMax program. That was not too hard was it?

Working With Colors

In the previous section we drew a WHITE rectangle on to a 640x480 graphic screen. Let us now draw a

different color rectangle. This is done via the **SetColor** instruction. This instruction allows us to select the ink color to draw with. Our new program now looks like this

```
Graphics 640,480,0
SetColor 255,0,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

Now build and run the above program. You will notice that instead of a white rectangle, we now have a red rectangle.

The **SetColor** instruction uses the RGB concept. RGB stands for Red, Green and Blue which are the 3 primary colors in the digital world. The actual color used for drawing will be the combination of the RGB primary colors.

Let us do a little diversion here and learn a little trick. If you hover your cursor on top of the SetColor command in the MaxIDE (see red circle) and then press the F1 key, you should see the required syntax of that command in the status bar (see red arrow)

Pressing F1 again (i.e. twice in total) will take you to the BlitzMax Help screen where there will be a brief explanation on how to use the function. You can also use the Navigation Bar on the right to scroll through the various functions available.

Notice how the **SetColor** instruction requires three parameters, the intensity of the red, green and blue colors as numbers between 0 to 255; 255 being the most intense. To obtain a RED color, we issue the instruction **SetColor 255,0,0 .** What we are saying here is that we want maximum intensity RED and zero intensity GREEN and zero intensity BLUE. Note that the intensity for each color can be set from zero to 255 (minimum to maximum intensity).

Let us experiment and see what color we get by combining a maximum intensity RED with a maximum intensity GREEN and zero intensity blue like this:-

```
Graphics 640,480,0
SetColor 255,255,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

Now build and run the above program. We now get a YELLOW rectangle which is the color we get when we mix RED with GREEN

Lets now play around with the intensity by reducing the RED component from 255 to 128. Note that we have also reduced the intensity of the GREEN color back to zero:-

```
Graphics 640,480,0
SetColor 128,0,0
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

Now build and run the above program. What we now get is a much subdued RED rectangle due to the reduced intensity

I hope that was clear that when we issue the **DrawRect** (or any other drawing command in BlitzMax), the color that will be used can be set by the **SetColor** command which takes three parameters; red intensity, green intensity and blue intensity. If the **SetColor** command is not issued, the color white will be used.

Introduction to Variables

Most BlitzMax instructions allow us to vary their behaviours. In the previous tutorial we learnt about the SetColor instruction. The ink color that it uses to draw the rectangle in our example above was based on the intensities of the defined RGB primary colors.

Instead of issuing the instructions SetColor 255,0,0 we can also issue the instructions like so:-

```
Graphics 640,480,0
R=255
G=0
B=0
SetColor R,G,B
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

If we were to run the above program, we will see that the same red rectangle which we have seen before will be drawn on the graphic screen

Note the 3 new lines R=255, G=0 and B=0. The instruction R=255 assigns or stores the number 255 into the variable R.

What are variables? We can think of variables as containers where numbers are stored in. In this example we have placed the numbers **255**, **zero** and **zero** into the 3 variables **R,G** and **B** respectively So when BlitzMax receives the instruction **SetColor R,G,B** it first retrieves the number from each variable and then carries out its color mixing.

Variables which stores numbers are called **integer** variables. Later we will learn about other different types of containers (or variables) that BlitzMax has.

Lets re-run the program by changing the assignment of the G variable

```
Graphics 640,480,0
```

```
R=255
G=255
B=0
SetColor R,G,B
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

Voila, what we now get is a bright yellow rectangle. Notice this is similar to saying **SetColor 255,255,0**

Assignment to variables can also be made differently. Let us look at this example

```
Graphics 640,480,0
R=255
G=R
B=0
SetColor R,G,B
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

If we were to run this we also get a yellow rectangle. The reason is when we issue the instruction R = 255, we placed the number 255 into the variable R.

When we issue the instruction G = R, we are telling BlitzMax to pick the number which is stored in the variable R and assign that same number to the variable G.

Lets now have a bit of fun and see what happens when we assign random numbers to the RGB variables.

```
Graphics 640,480,0
SeedRnd MilliSecs()
R = Rnd(255)
G = Rnd(255)
B = Rnd(255)
SetColor R,G,B
DrawRect 50,50,200,100
Flip
```

```
WaitMouse
End
```

We've introduced two new instructions here, **SeedRnd Millisecs()** and **Rnd(255)**.

The instruction **Rnd(255)** simply creates a random number between 0 and 255. This random number is then assigned to each of our RGB variables.

In order for our **Rnd(255)** instruction to work properly, we have first to issue the instruction **SeedRnd Millisecs()**. At this stage let us not worry about why this is the case yet. The important thing to remember is that some instructions depends on other instructions to work properly.

Go on and run this program a few times. Each time we run it we get a different colored rectangle on the graphic screen.

The different colors are caused by different numbers randomly generated by the RND(255) instructions each time we run our program

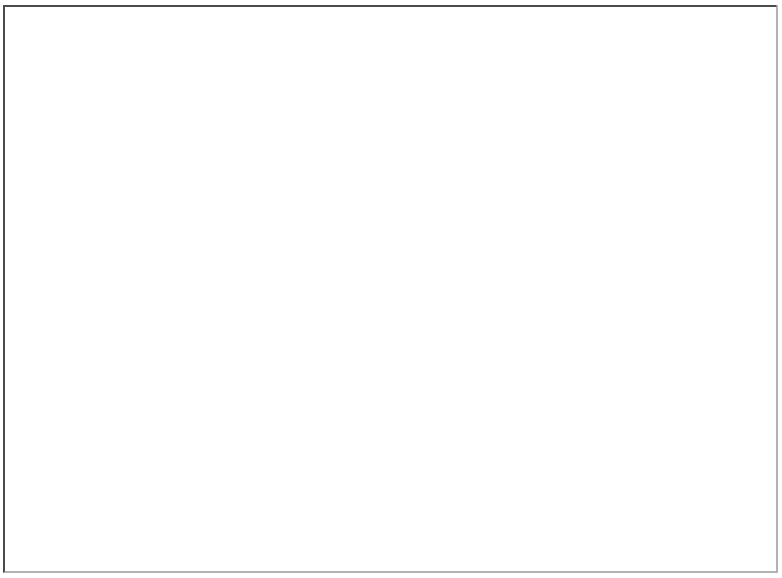
Let us now familiarize ourselves with another type of BlitzMax variable, the string variable. Earlier we saw that variables which stores numbers are called integer variable. String variables stores text characters (also known as strings) instead of numbers.

To help differentiates between integer variables and string variables, we have to post-fix our variable name by the $ sign.

Let us see an example of this

```
Graphics 640,480,0
a$="Hello World"
DrawText a$,10,20
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

If we compile and run the above program we get the following:-



The instruction **a$="Hello World"** assigns the phrase \93Hello World\94 to the string variable a$. Strings within BlitzMax must be enclosed within the quote marks.

The next instruction **DrawText a$,10,20** writes the text within the string variable a$ to the graphic screen at the screen coordinates 10,20

Now lets do something useful with this new knowledge and modify our random color program like this

```
Graphics 640,480,0
SeedRnd MilliSecs()
R = Rnd(255)
G = Rnd(255)
B = Rnd(255)

A$ = "RGB COLOR "+R+","+G+","+B
DrawText a$,10,20

SetColor R,G,B
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

If we compile and run the above program we get the following:-

The above picture tells us that the combination of the primary colors R,G,B of intensities 93, 86, 119 produces a rectangle with a greyish, purplish color. You will probably get a different color on your screen.

Note that the assignment to the string variable A$ in this example is now more complicated then before. Let us try and dissect the line.

```
Graphics 640,480,0
SeedRnd MilliSecs()
R = Rnd(255)
G = Rnd(255)
B = Rnd(255)

A$ = "RGB COLOR "+R+","+G+","+B
DrawText a$,10,20

SetColor R,G,B
DrawRect 50,50,200,100
Flip
WaitMouse
End
```

The plus sign (+) that we see above allow us to concatenate (add) multiple strings together.

For example if we have the assignments **A$=\94MY \93** and **B$=\94CAT\94**

Then an assignment **C$=A$+B$** will result in the string \93MY CAT\94 being stored in the string variable C$.

So what we are doing with this assignment **A$ = "RGB COLOR "+R** is assembling the string \93RGB COLOR \93 plus the number stored in the variable R. To make our displayed text to look nicer we also add in the comma before we assemble the next number i.e. **A$ = "RGB COLOR "+R+\94,\94**

We then go on and assemble the G and B parts resulting in the final line ;-

```
A$ = "RGB COLOR "+R+","+G+","+B
```

Note that BlitzMax makes its easy for us by allowing us to mix numbers and strings in this manner.

Introduction to Loops

Its not that much fun having to run our program each time we want to see a different colored rectangle. Lets see whether BlitzMax can help us out here. In order to do this, we need to be introduced to the concept of loops.

So far what we have done was to issue instructions to BlitzMax in a linear fashion. BlitzMax executes each instruction one line at a time until it reaches the END instruction upon which BlitzMax then stops. This is also known as a linear Program Flow.

Lets see how a loop will work in BlitzMax

```
Graphics 640,480,0
SeedRnd MilliSecs()
R = Rnd(255)
G = Rnd(255)
B = Rnd(255)
```

```
Repeat

    R = Rnd(255)
    G = Rnd(255)
    B = Rnd(255)

    SetColor R,G,B
    DrawRect 50,50,200,100
    Flip

Until MouseHit(1)

End
```

What we now see when we run the above program is a flashing rectangle. This is because the loop is going through too fast. We will fix this later.

We now have 2 new instructions to look at**, Repeat** and **Until MouseHit(1)**

```
Repeat

    R = Rnd(255)
    G = Rnd(255)
    B = Rnd(255)

    SetColor R,G,B
    DrawRect 50,50,200,100
    Flip

Until MouseHit(1)
```

When BlitzMax encounters the **Repeat** instruction, it knows that it is now entering the start of a loop. BlitzMax will then executes the other instructions line by line as before.

When BX gets to the **Until MouseHit(1)** instruction, it will check to see whether we have click the left mouse button. If we have not, it will then loop back to the line which contains the **Repeat** instruction and continues to perform the instructions in the subsequent lines below it.

This LOOP will repeat infinitely until we click the mouse button upon which it will exit the loop and then continues on to the next line which in our particular example happens to be the instruction to **End** the program

Now lets do something about that flashing rectangle by introducing another instruction called **Delay**

```
Graphics 640,480,0
SeedRnd MilliSecs()
R = Rnd(255)
G = Rnd(255)
B = Rnd(255)

Repeat

    R = Rnd(255)
    G = Rnd(255)
    B = Rnd(255)
```

```
    SetColor R,G,B
    DrawRect 50,50,200,100
    Flip
    Delay(500)

Until MouseHit(1)

End
```

The **Delay (500)** instruction tells BlitzMax to pause our program for 500 milliseconds (which is half a second) before going to the next line

Moving an object on the Graphic Screen

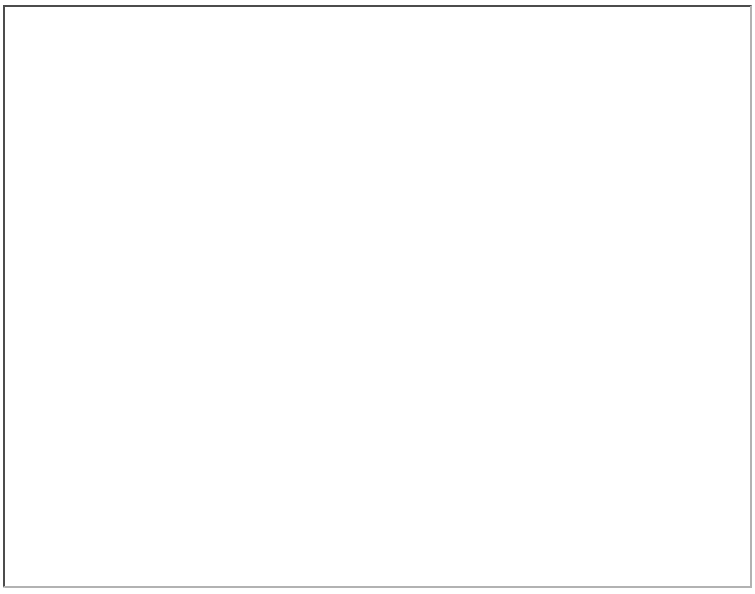
In games, we normally see all sorts of objects whizzing around the screen. Lets now make a start to move our rectangle around the graphic screen.

Lets go back to our simple white rectangle but lets introduce two new integer variables X and Y

```
Graphics 640,480,0
X=50
Y=50
Repeat
   A$ = "RECTANGLE COORDINATES "+X+","+Y
   DrawText A$, 10,25
   DrawRect X,Y,200,100
   Flip
Until MouseHit(1)

End
```

We can see that the values of the integer variables X and Y is now displayed on the screen with our familiar white 200x100 rectangle.



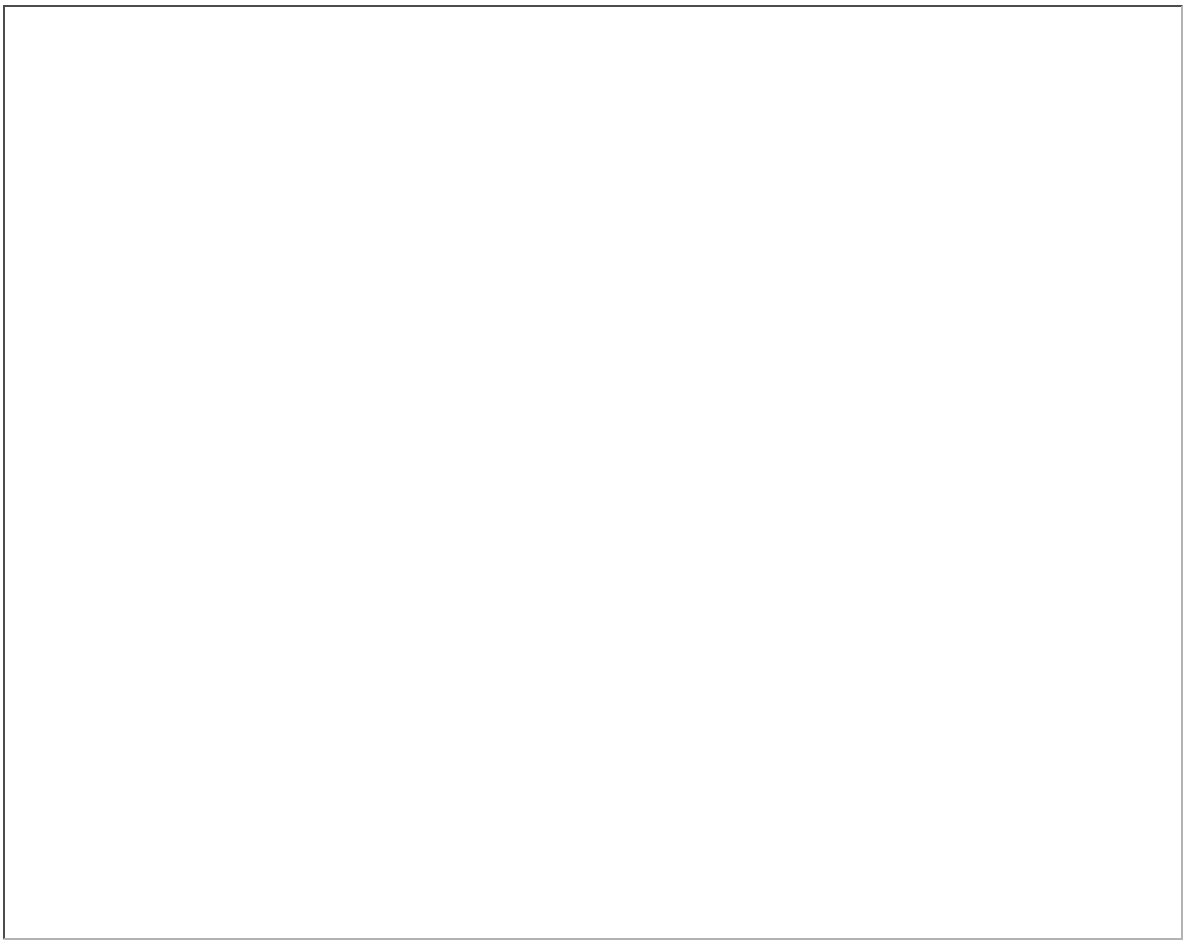
We are now going to introduce another concept, the concept of **incrementing** our variables. Let us add one new instruction to the previous program

```
Graphics 640,480,0
```

```
X=50
Y=50
Repeat
    X :+ 1
   A$ = "RECTANGLE COORDINATES "+X+","+Y
   DrawText A$, 10,25
   DrawRect X,Y,200,100
   Flip
Until MouseHit(1)

End
```

Building and running the above program yields an ever growing white rectangle.



The reason this is the case is because the number stored in the variable X is increasing by 1 each time we go through the loop. This behavior is achieved by the instruction

```
X :+ 1
```

The above instruction tell BlitzMax to increment the number stored in the integer variable X by 1. So the first time round, X will contain the number 51 as initially we stored 50 into the integer variable X by our **`X=50`** instruction.

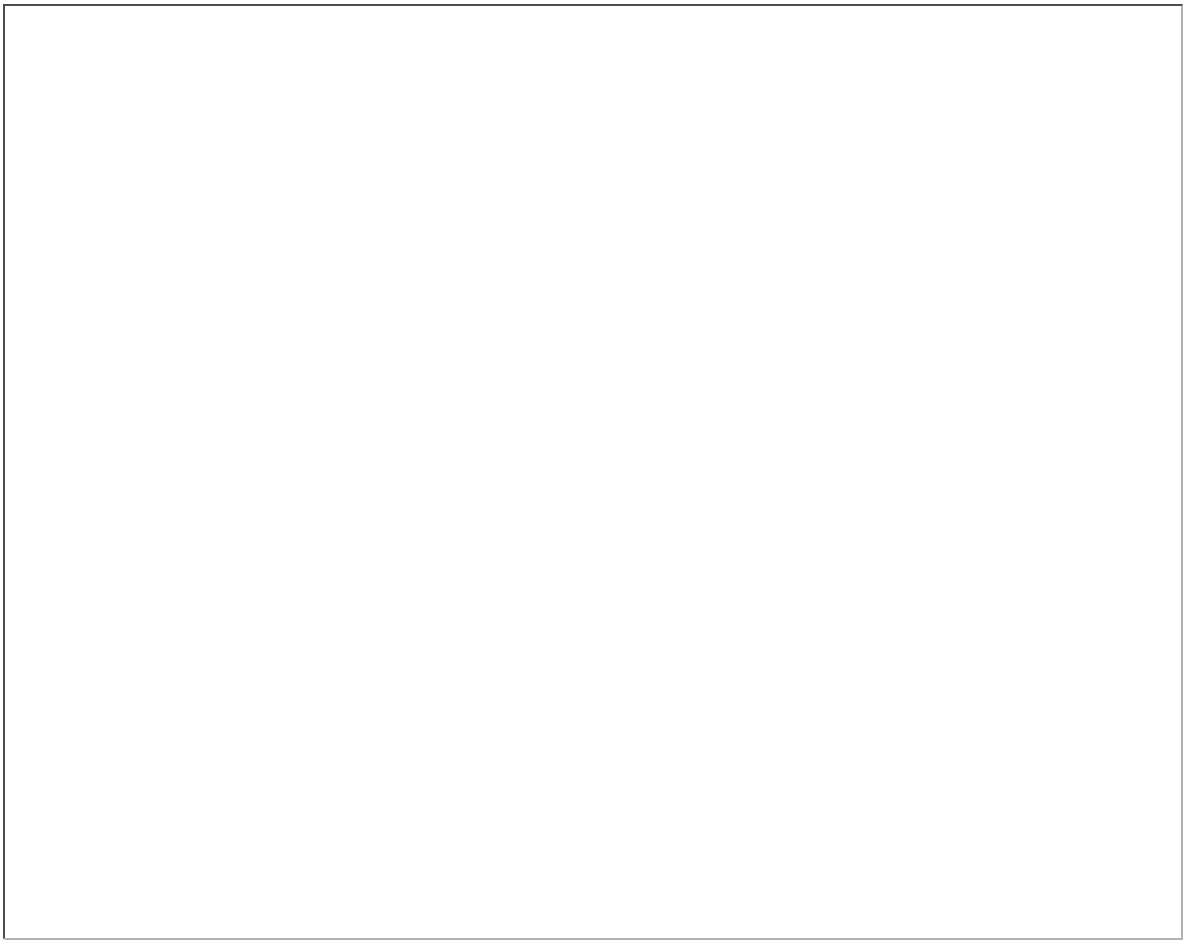
The start of our rectangle is then moved one pixel to the right as well. As we move around the loop, the rectangles get drawn on top of each other, thus the rectangle appears to grow longer rather than moving.

Now lets make a small modification to our program by adding the **`CLS`** (Clearscreen) instruction and then see what happens.

```
Graphics 640,480,0
X=50
Y=50
Repeat
    Cls
    X :+ 1
   A$ = "RECTANGLE COORDINATES "+X+","+Y
   DrawText A$, 10,25
   DrawRect X,Y,200,100
   Flip
Until MouseHit(1)

End
```

Notice that instead of growing bigger, the rectangle is now actually moving to the right. We can also see the Rectangle Coordinates incrementing as per our **DrawText** instruction.



Remember in the beginning of this tutorial we came across this curious fact that the **DrawRect** instructions draws to a hidden area and we had to **Flip** in order to see the rectangle?

When we issue the **Cls** instruction, the hidden area (also known as the BACKBUFFER) is first cleared of all previous drawings thus allowing us to clearly see the rectangle moving instead of being drawn over the previous drawings (albeit moving slightly to the right each time)

So there we have it, a moving object on our graphic screen. These set of instructions (**Repeat Until** Loop, **Cls**, Drawing and **Flip**) forms the basis of most game programs

In later tutorials we will learn about two other elements, USER INPUT to control PLAYER Movement and ARTIFICIAL INTELLIGENCE to control ENEMY Movement

But first, we need to be introduced to the conditional **IF** statement

The IF statement

The IF statement is a class of programming concept called conditionals. Let us see how to use this in practice.

Lets us insert an IF statement into our program and also make our rectangle move faster by incrementing with a bigger number

```
Graphics 640,480,0
X=50
Y=50
Repeat
    Cls
    X :+ 1
    If X>400 Then X=0
   A$ = "RECTANGLE COORDINATES "+X+","+Y
   DrawText A$, 10,25
   DrawRect X,Y,200,100
   Flip
Until MouseHit(1)

End
```

We now see our rectangle moving much faster on the screen as well as moving back to the left side once it reaches the far right of the screen.

This re-start back to the left side is achieved by our new instruction **`IF X>400 THEN X=0`**

The IF instruction works by evaluating the logic of **`X>400`** (read as X greater than 400**)**. The expression **`X>400`** is a true statement when the number contained in the X variable is bigger than 400. If this statement is true, the instruction after THEN is then carried out i.e the value zero is stored into the integer variable X.

If this statement is false, then the instruction **`X=0`** is not carried out and the variable X gets to retain whatever value it held before.

In later tutorials we will be introduced to more complicated forms of the IF statement. The IF statement (or more accurately conditionals) is the cornerstone of programming logic.

Reacting to User Input

In the previous section, we learnt how to move a rectangular object on the screen own its own accord. We simply issue the right set of instructions (programming logic) to move the object to the far right and back again.

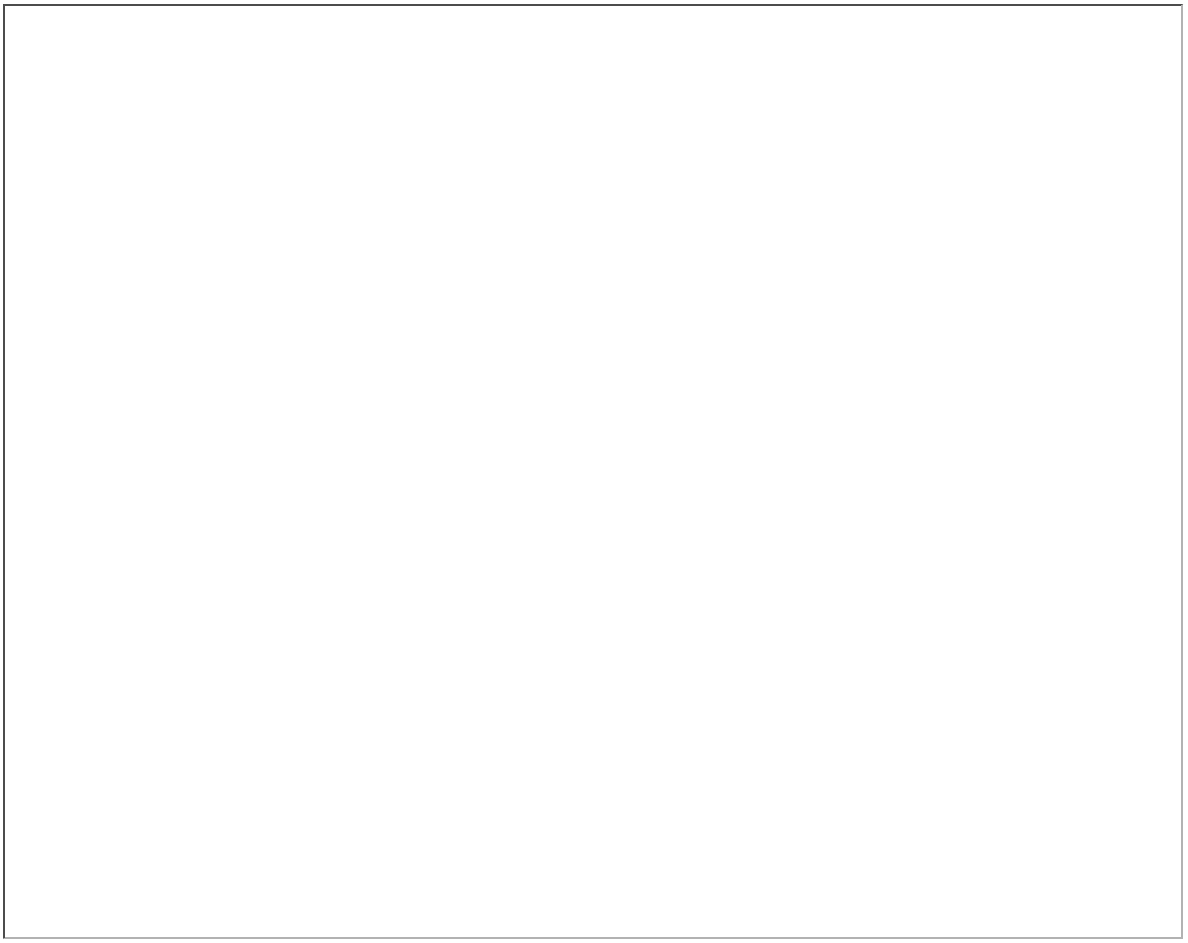
Let us now see how we can control the movement of the object using our mouse. We will now replace two lines from the above program with two new functions, the **MouseX**() and **MouseY**() functions.

```
Graphics 640,480,0
X=50
Y=50
Repeat
    Cls
```

```
    X = MouseX()
    Y = MouseY()
   A$ = "RECTANGLE COORDINATES "+X+","+Y
   DrawText A$, 10,25
   DrawRect X,Y,200,100
   Flip
Until MouseHit(1)

End
```

Now build and run the above program. You should now be able to move the rectangle by moving the mouse around the screen..



This behaviour is achieved by the introduction of two new instructions **`MOUSEX()`** and **`MOUSEY().`**

What **MouseX()** does is it returns the X coordinate of the mouse and the X=**MousxeX**() instruction then assigns this returned coordinate to the X integer variable. Similarly for the **MouseY**() instruction.

What we have learnt so far

We have learnt some fairly fundamental stuff about programming today

- We learnt how to compile a BlitzMax program and to execute it. With BlitzMax this is done easily by clicking the "Build and Run" button..
- We've learnt about how the instructions in programs are executed sequentially unless told otherwise
- We've learnt about variables; storage areas to store numbers and text
- We've learnt about loops, where series of the same instructions gets repeated
- We've learnt about conditionals, using the IF statement to only execute an instruction when the

conditions are right
- We've learnt how to display graphic objects onto a graphic screen
- We've learnt how users can interact with our program using the mouse
- We've learnt how to make our graphic object moves on the graphic screen
- We've learnt how to use the help file

Blitzmax commands introduced so far

So even with our first tutorial we have covered quite a lot of new stuff. Don't worry if the full syntax of the functions below looks complicated. The simple usage are as per shown above:-

Function **Print**(str$="")
Write a string to the standard IO stream.

Function **Graphics**:TGraphics(width,height,depth=0,hertz=60,flags=0)
Create a graphics object.

Function **DrawRect**(x#,y#,width#,height#)
Draw a Rectangle at position x,y of size width and height

Function **Flip**(sync=-1)
Swaps the front and back buffers of the current graphics objects.

Function **WaitMouse**()
Wait for mouse button click.

Function **SetColor**(red,green,blue)
Set current color.

Function **DrawText**(t$,x#,y#)
Draw text (in t$) at location x,y

Function **WaitMouse**()
Wait for mouse button click.

Function **MouseX**()
Get mouse x location.

Function **MouseY**()
Get mouse Y location.

Function **Rnd!(** min_value!=1,max_value!=0)
Generate random double.

Function **RndSeed**()
Get random number generator seed.

Keyword **If**

Begin a conditional block.

Keyword **End**

End program execution.

Function **Delay**(millis)

Delay suspends program execution for at least **millis** milliseconds.
A millisecond is one thousandth of a second.

Back to the Main Index