

Make a Game with Small Basic

Getting Small Basic

- Go to the Windows app store and search for “Small Basic” and click the Install button *or*
- Go to www.smallbasic.com and click the download button
- Then select your language and click download and follow the installation instructions!



Lesson 1: Tips & Tricks

#1 Pro-Tip: BE LAZY!

- Start typing and hit tab or enter to complete the code
- Use the up and down arrow buttons to scroll through the methods
- Look at the explanations on the right-hand side

Lesson 2: The Screen

We're going to make a simple game where you have to "bounce" a ball around the screen. To do this, we're going to use the `GraphicsWindow` object. Let's start with a really simple program.

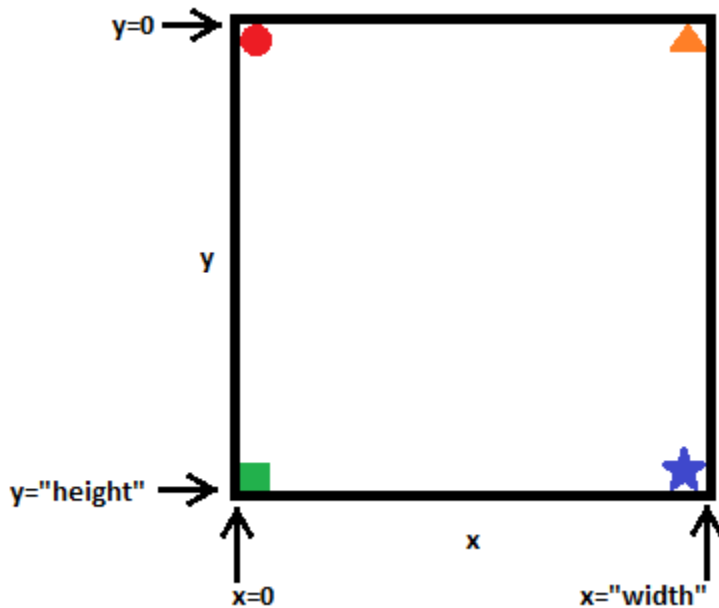
```
GraphicsWindow.BackgroundColor = "DarkBlue"
```



Click the `Run (F5)` button at the top. What does it do?

Before we go on, you should know a little bit about this `GraphicsWindow`. Small Basic uses coordinates to figure out where things go. The `x` position is how far left or right an object is and the `y` position is how far up or down an object is.

If you want an object to be on the screen, its `x` value must stay between 0 and the screen's width and its `y` value must stay between 0 and the screen's height.



Here's an example of some shapes at different places on the screen.


Shape	x	y
Red circle	0	0
Orange triangle	<code>GraphicsWindow.Width</code>	0
Green square	0	<code>GraphicsWindow.Height</code>
Blue star	<code>GraphicsWindow.Width</code>	<code>GraphicsWindow.Height</code>

Lesson 3: Getting set up

We need at least two things to play this game: a ball to bounce, and a paddle to bounce it with. Let's start with the paddle.

```
paddle = Shapes.AddRectangle(120, 12)
```



Click the  button at the top. What does it do?

Okay, we have a paddle! We told it to be 120 wide and 12 tall. But it's not where we want it to be. We want to put it at the bottom of the screen. What do we set x and y to in order to make it go to the bottom of the screen?

We set it to the "height" of the screen, minus the height of the paddle. That way the whole paddle fits on the screen.

```
Shapes.Move(paddle, 0, GraphicsWindow.Height - 12)
```

Now it's at the bottom left corner of the screen! We also need a ball. Let's draw that.

```
ball = Shapes.AddEllipse(16, 16)
```

Want to make your shapes a little different? Here are some things you can try! Where do you think you would put them in the code?

Change Ball Color:

Add this code before the ball is drawn to change what color it is:

```
GraphicsWindow.BrushColor = "Red"  
GraphicsWindow.PenColor = "White"
```

Change Background Color:

```
GraphicsWindow.BackgroundColor = GraphicsWindow.GetRandomColor()
```

✍ CHALLENGE QUESTION: How would you change the ball color to a random color?

✍ CHALLENGE QUESTION: How would you make the paddle smaller?

Lesson 4: Let's get moving!

The paddle should move when the mouse moves. How can we do this?

First, we need a **subroutine**. Normally, all the code we type in is run from top to bottom and then the program stops. If we want to run some parts of the code many times, one of the ways we can do this is by putting it in a **subroutine**.

Every time the mouse moves, the paddle should move left or right with it. We can get the mouse position from `GraphicsWindow`, and we can move the paddle using `Shapes.Move`. Where do we want to move the paddle to?

```
Sub WhenMouseMoves
```

```
    mouseX = GraphicsWindow.MouseX
```

```
    Shapes.Move(paddle, mouseX-60, GraphicsWindow.Height-12)
```

```
EndSub
```

You might be wondering what mouseX means. The word “mouseX” is called a **variable**. A variable is a place where you can store a little piece of information to use in your program. We use this to remember where the paddle is supposed to move.

Wait, we forgot to actually run the subroutine! The `GraphicsWindow` has some events that might help us. Try to guess which one we need by typing `GraphicsWindow` and pressing the arrow keys to go through the properties. Once you find it, add it **above** our subroutine.

```
GraphicsWindow.MouseMove = WhenMouseMoves
```

This code means that every time the mouse moves, our `WhenMouseMoves` subroutine should



run. Click the `Run (F5)` button to try it!

Lesson 5: Making the ball move

Now that we have our paddle, we want to make the ball bounce around the screen too. We're going to use some variables to keep track of everything. First, we use "x" and "y" to keep track of where the ball is right now. Add this code after `EndSub`.

```
x = 0
```

```
y = 0
```

To start with, we want the ball to go a little to the right and a little bit down.

```
x = x + 1
```

```
y = y + 1
```

```
Shapes.Move(ball, x, y)
```



Click the `Run (F5)` button at the top. What does it do?

The ball moved, but barely. We want it to keep moving forever- let's use a **while loop** to do this. Remember, normally all the code we type in is run from top to bottom and then the program stops. A **loop** will make the program jump back to a certain place and run the code again and again until the looping stops.

A while loop will run forever as long as the code right after `While` is something true. We'll keep track of this in a **variable**- if `stillPlaying` is true, we'll keep running the loop. When we want to end the loop later, we'll change `stillPlaying` to something besides “True” so the loop will stop. The new code should look like what's on the next page (There are comments, but you don't have to type those in your program)

```

x = 0
y = 0
stillPlaying = "True" '-- the variable we're going to use to keep the While loop going
While stillPlaying = "True" '-- as long as this statement is true, we'll continue
    x = x + 1
    y = y + 1
    Shapes.Move(ball, x, y)
EndWhile '-- this is the end of the section of code we want to repeat

```

Now the code should run, move the ball a little, and jump back up to **While** and move again.



Click the **Run (F5)** button at the top. What does it do?

I didn't even see the ball move, but it's gone. Computers run *really* fast, so what happened was that it moved the ball all the way off the screen before you could notice it. Let's make it slow down by adding a small delay before the **EndWhile**.

```

Program.Delay(5)

```



Click the **Run (F5)** button at the top and see what happens to the ball now.

🔪CHALLENGE: How would you make the ball move slower?

We should show a message when the ball goes off the bottom of the screen, because the point of the game is to keep the ball bouncing! Before we make the program jump back to the start of the **While** loop, let's check if the ball is still on the screen. And if it's not we should show a message saying the game is over and then make the **While** loop stop.

How can we tell if the ball is past the bottom of the screen?

```

If (y > GraphicsWindow.Height) Then
    GraphicsWindow.ShowMessage("You Lose", "Paddle")
    stillPlaying = "False"
EndIf

```



Click the **Run (F5)** button at the top and see what happens when the ball goes off screen.


Lesson 6: Make it bounce!

Our program isn't really a game yet- it's missing some important features. First of all, the ball doesn't actually bounce off the paddle! It just moves down and to the right. To make it easier to change the ball's direction, we want to use **variables** for the ball's movement instead of just adding 1 every time.

You should put the new variables before the **While** label, we don't want to keep setting them every time the loop runs!

```
moveX = 1
moveY = 1
stillPlaying = "True"
While stillPlaying = "True"
    x = x + moveX
    y = y + moveY
```



Click the  button at the top, just to make sure that this didn't change anything.

⚡CHALLENGE: What happens if you make moveX or moveY bigger?

Now it's time to make the ball bounce off the paddle. First, we have to find out where the paddle is! We know it's at the bottom of the screen, but it's constantly moving left or right so we need to use **Shapes** to figure out where it is right now.

Try scrolling through the methods on **Shapes** to figure out how we can get the x value for the paddle, and put that code right after the **While** line .

```
paddleX = Shapes.GetLeft(paddle)
```

We need to check three things to make sure the ball is on the paddle:

- 1) The ball has to be at the bottom of the screen
- 2) The ball has to be further right than the left edge of the paddle
- 3) The ball has to be further left than the right edge of the paddle


How would we write each of these in code? Remember, the paddle is **120** wide and **10** tall.

And if the ball is on the paddle, what do we want to make it do? We want to make it start moving **up** instead of **down**. What can we change to make it start moving upwards? (Hint- it involves one of the move variables!)

This is how the code should look to make the ball bounce off the paddle:

```
If (y >= GraphicsWindow.Height - 10 and x >= paddleX and x <=
paddleX + 120) Then
    moveY = -moveY
EndIf
```



Click the  button at the top to try it out!

It looks like two things went wrong. First, it looks like the ball bounced off the screen instead of off the paddle. That's because when we checked if the ball was at the bottom of the screen, we did this:

```
y >= GraphicsWindow.Height - 10
```

But x and y are the coordinates for the *top left* corner of the ball! So we were making the ball bounce once the *top* of it hit the paddle, which doesn't look right. Change the **10** into a better value. Remember, the ball is **16** tall. Keep running your program until the bounce looks right.

Secondly, the ball bounced right off the screen! We need to make sure it doesn't leave the screen on accident. Luckily, we know exactly where the ball is and where it's not supposed to go. We want to make sure it:

- 1) Doesn't go too far up
- 2) Doesn't go too far right
- 3) Doesn't go too far left

We already handle what happens if it goes too far down. It either hits the paddle or you lose!

We can check the first condition really easily. We know the top of the screen is where y is zero, so if our ball's y position is zero or less we need to bounce it. Just like we did for the paddle bounce, we need to reverse the moveY direction.

```
If (y < 0) Then
    moveY = -moveY
EndIf
```

How do we check the other two conditions? What do we need to change to make it bounce off the sides? Try to figure it out yourself! Remember, x and y tell you where the *top left* corner of the ball is!

Here's the code. We have to subtract 16 from the width because the ball is 16 wide, and we want to bounce when the right side of it hits the wall.

```
If (x >= GraphicsWindow.Width - 16 or x < 0) Then
    moveX = -moveX
EndIf
```

And we're done! We now have a completed game!


✂ CHALLENGE: Can you make this game harder? Easier?

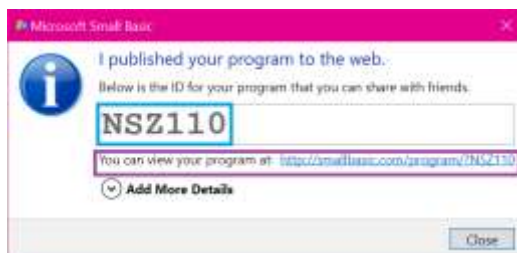
✂ CHALLENGE: Can you add shapes that disappear when the ball hits them?

✂ CHALLENGE: Something weird happens if the ball hits the edge of the paddle- can you fix it? ✂ CHALLENGE: Can you add another ball (difficult)?

Lesson 7: Share your code!

In Small Basic, you can export a program so that other people with Small Basic can look at the code, run it and edit it.


Simply click the  button in the top menu. You'll get a popup that looks like this:



The special code in blue is what you can share with your friends who also use Small Basic. The link in purple is what you can share with friends and family who don't write code; you can email or text it to them!

Lesson 7: Learn from others!

In Small Basic, you can import someone else's code to work on it and fix a problem, make it better, or just customize it for fun! It's also a good way to learn how others write code and learn from their examples. We're going to import in a Tetris game and then play with the code.

Click  and type in "tetris" (all lowercase). The Tetris code appears!

✂ CHALLENGE: What changes can you make to the Tetris game?