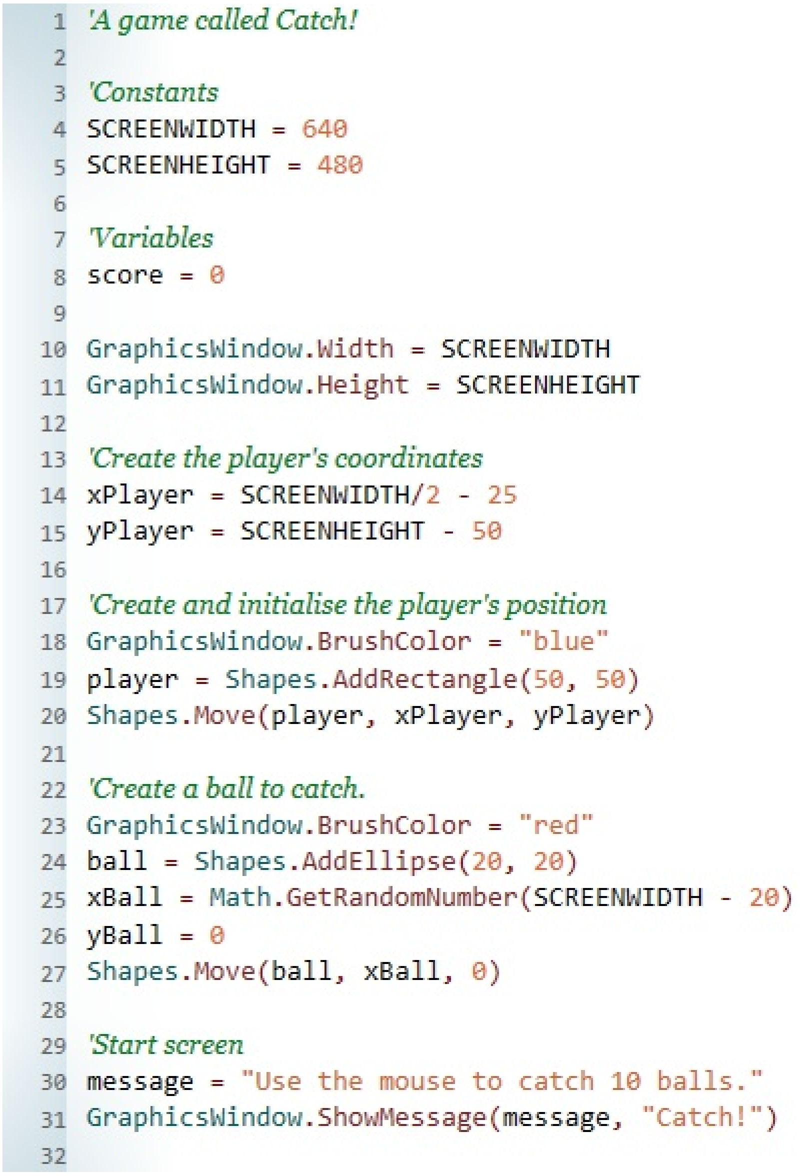
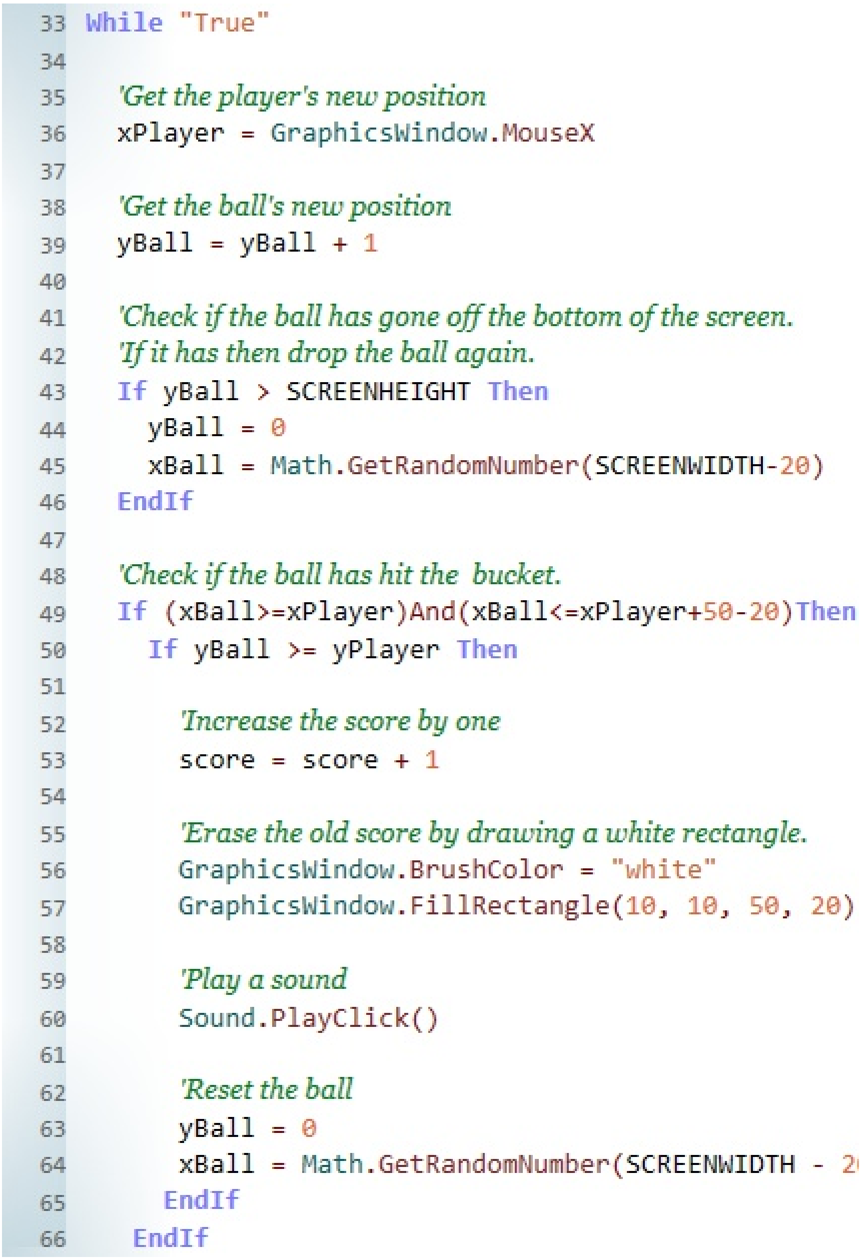
Making a simple game

In this section we will make a simple catching game. It will use what we already know and a couple of extra objects from the Small Basic library. The program will be longer than anything that we have written so far. After the program listing I will explain each section in detail.



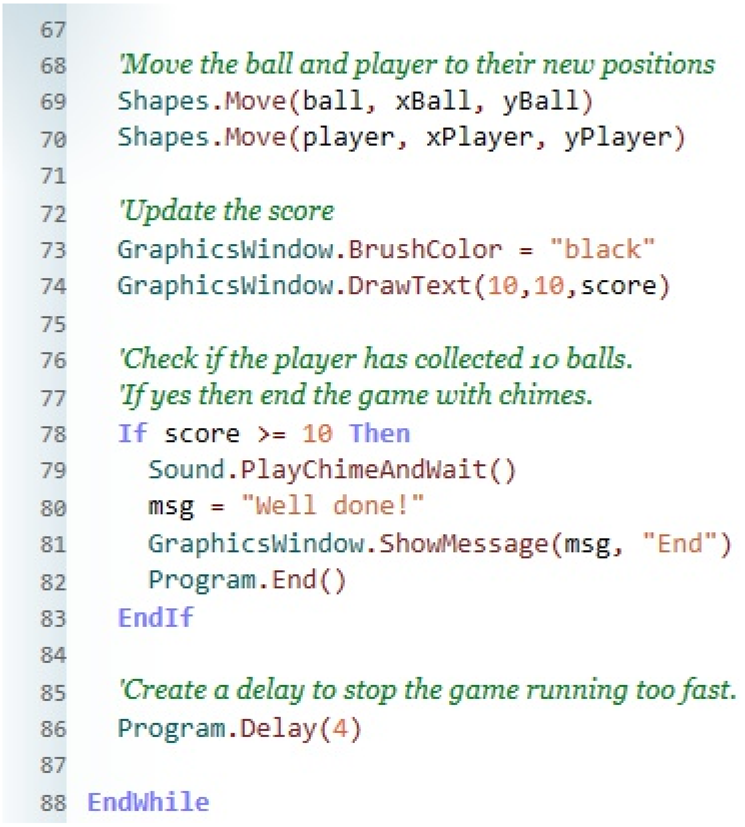
```
'A game called Catch!

'Constants
SCREENWIDTH = 640
SCREENHEIGHT = 480

'Variables
score = 0

GraphicsWindow.Width = SCREENWIDTH
GraphicsWindow.Height = SCREENHEIGHT

'Create the player's coordinates
xPlayer = SCREENWIDTH/2 - 25
yPlayer = SCREENHEIGHT - 50

'Create and initialise the player's position
GraphicsWindow.BrushColor = "blue"
player = Shapes.AddRectangle(50, 50)
Shapes.Move(player, xPlayer, yPlayer)

'Create a ball to catch.
GraphicsWindow.BrushColor = "red"
ball = Shapes.AddEllipse(20, 20)
xBall = Math.GetRandomNumber(SCREENWIDTH - 20)
yBall = 0
Shapes.Move(ball, xBall, 0)

'Start screen
message = "Use the mouse to catch 10 balls."
GraphicsWindow.ShowMessage(message, "Catch!")

```

Catch listing continued



```
While "True"

  'Get the player's new position
  xPlayer = GraphicsWindow.MouseX

  'Get the ball's new position
  yBall = yBall + 1

  'Check if the ball has gone off the bottom of the screen.
  'If it has then drop the ball again.
  If yBall > SCREENHEIGHT Then
    yBall = 0
    xBall = Math.GetRandomNumber(SCREENWIDTH-20)
  EndIf

  'Check if the ball has hit the  bucket.
  If (xBall>=xPlayer)And(xBall<=xPlayer+50-20)Then
    If yBall >= yPlayer Then

      'Increase the score by one
      score = score + 1

      'Erase the old score by drawing a white rectangle.
      GraphicsWindow.BrushColor = "white"
      GraphicsWindow.FillRectangle(10, 10, 50, 20)

      'Play a sound
      Sound.PlayClick()

      'Reset the ball
      yBall = 0
      xBall = Math.GetRandomNumber(SCREENWIDTH - 2
     EndIf
   EndIf
```

Catch listing continued



```

  'Move the ball and player to their new positions
  Shapes.Move(ball, xBall, yBall)
  Shapes.Move(player, xPlayer, yPlayer)

  'Update the score
  GraphicsWindow.BrushColor = "black"
  GraphicsWindow.DrawText(10,10,score)

  'Check if the player has collected 10 balls.
  'If yes then end the game with chimes.
  If score >= 10 Then
    Sound.PlayChimeAndWait()
    msg = "Well done!"
    GraphicsWindow.ShowMessage(msg, "End")
    Program.End()
  EndIf

  'Create a delay to stop the game running too fast.
  Program.Delay(4)

EndWhile
```

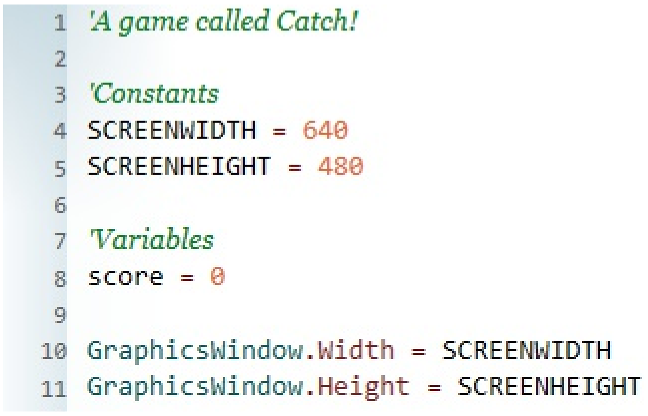
Before I explain how the program works, type it in and play it a few times. If you get any errors see if you can find them and remember that Small Basic will tell you (at the bottom of the editor) on what line there is an error. If you compare with the listing you should be able to find the mistake.

On the next page I will explain each of the commented sections.

Constants can are like variables but their values can not change. Programmers usually use Capitals to name them. Small Basic does not allow the programmer to create their own constants, but here we can pretend because the screen's width and height will not change in the code.

The game explained

The first part of the program starts with these lines.



```
1  'A game called Catch!
2
3  'Constants
4  SCREENWIDTH = 640
5  SCREENHEIGHT = 480
6
7  'Variables
8  score = 0
9
10 GraphicsWindow.Width = SCREENWIDTH
11 GraphicsWindow.Height = SCREENHEIGHT
```

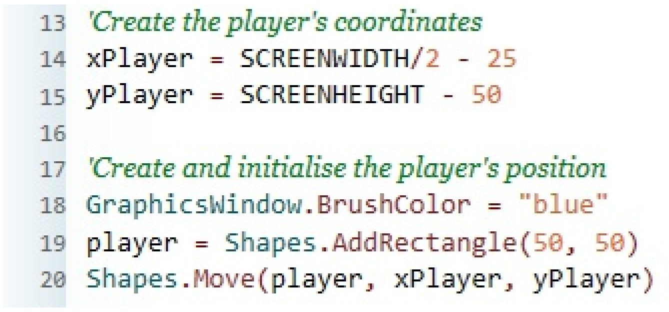
Line 1 is just a comment, I won't talk about these anymore since they are ignored by the computer.

Lines 4 and 3 assign two variables SCREENWIDTH and SCREENHEIGHT. They are written in capitals because I am treating them as constants; I do not expect the values of screen size to change. However, if I ever need to rewrite this program for screens of different sizes, I will only need to change the code here, not other places thoughout the program. So in the long run it could save me time.

Line 8 creates a score variable which is intialised to zero.

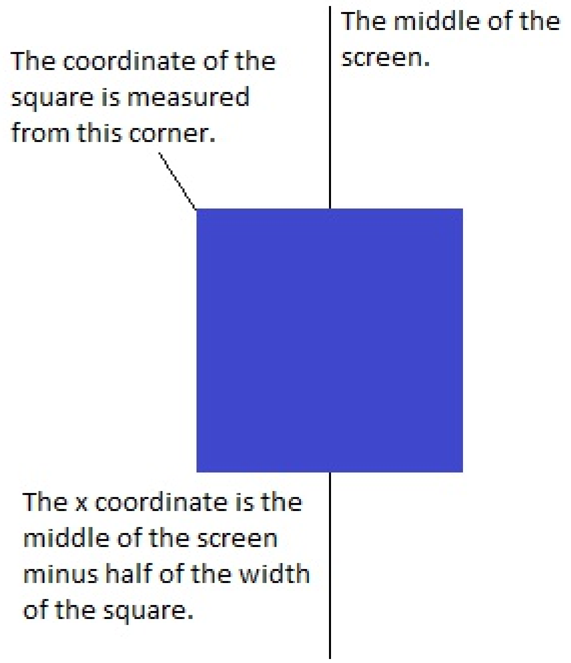
Lines 10 and 11 set the size of the Graphics Window to the values stored in the constant variables SCREENHEIGHT and SCREENWIDTH.

Game continued



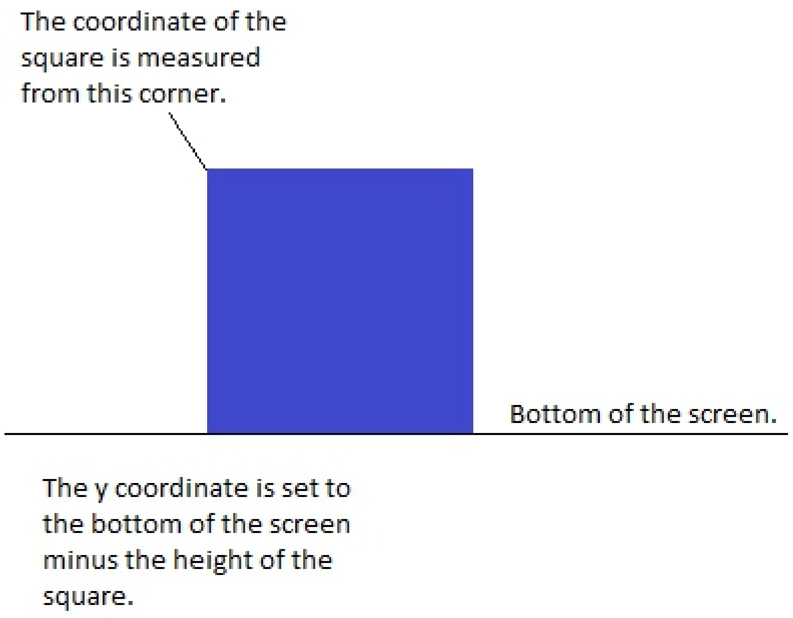
```
13 'Create the player's coordinates
14 xPlayer = SCREENWIDTH/2 - 25
15 yPlayer = SCREENHEIGHT - 50
16
17 'Create and initialise the player's position
18 GraphicsWindow.BrushColor = "blue"
19 player = Shapes.AddRectangle(50, 50)
20 Shapes.Move(player, xPlayer, yPlayer)
```

Lines 14 and 15 intialises the coordinates of the player, which will be a plain square shape. The first variable xPlayer is assigned the value SCREENWIDTH/2 -25 and will place the player in the middle of the width of the screen. We need the -25 part because the width of the player will be 50 pixels and the origin of the player is always the top left hand corner. So subtracting half of the player's width will place it in the exact middle.


The middle of the screen.
The coordinate of the square is measured from this corner.
The x coordinate is the middle of the screen minus half of the width of the square.

Unlike in mathematics, screen coordinates start in the top left corner. So the vertical value increases from top to bottom as it goes down the screen.

The y coordinate of the player is set to the bottom of the screen minus its height of 50 pixels, that way the player sits exactly on the bottom of the screen.

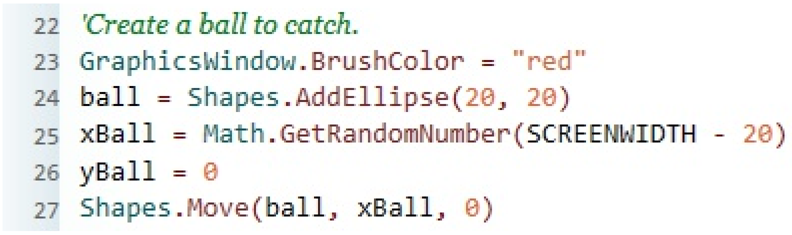

The coordinate of the square is measured from this corner.
Bottom of the screen.
The y coordinate is set to the bottom of the screen minus the height of the square.

Methods are just functions that are associated with particular objects.

The lines 18 to 20 create the player object, a blue square. To create a blue square we need to first set the GraphicWindow's BrushColor to blue. Then we use Small Basic's Shape object's AddRectangle(width, height) method to create a 50 pixel by 50 pixel square. The newly created square is given the name player by assigning it to the variable player. Now we can use the Move(shape name, x, y) method to move the player's square to its starting position.

The Shape object has lots of useful methods like Move(shape name, x, y), Rotate(shape name, angle) and Zoom(shape name, x scale, y scale).

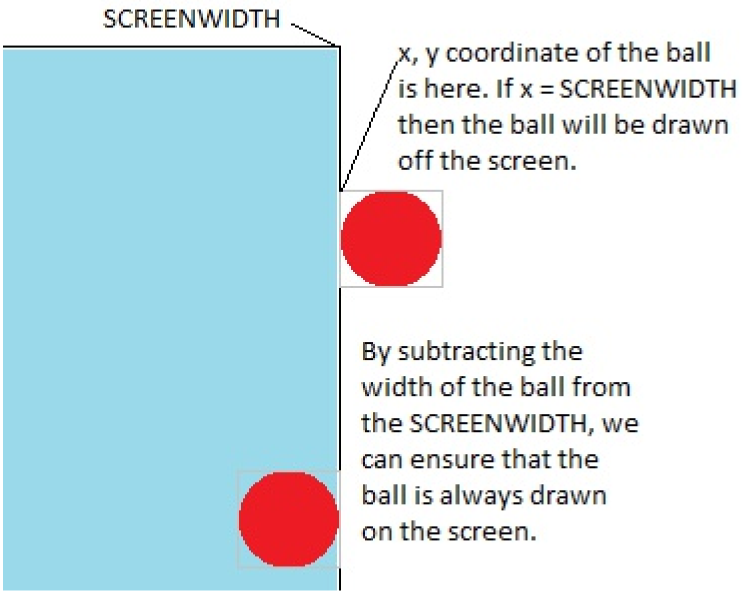
Game continued



```
22 'Create a ball to catch.
23 GraphicsWindow.BrushColor = "red"
24 ball = Shapes.AddEllipse(20, 20)
25 xBall = Math.GetRandomNumber(SCREENWIDTH - 20)
26 yBall = 0
27 Shapes.Move(ball, xBall, 0)
```

Lines 23 to 27 create a red ball in exactly the same way as we created the blue player object in the previous lines, though this time the shape is named ball.

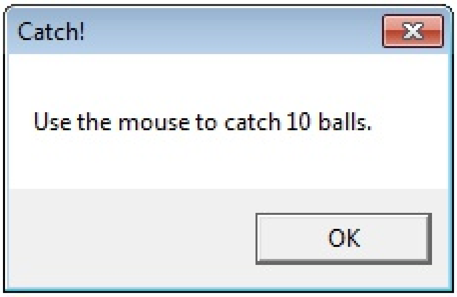
We want our ball to fall from a random position somewhere along the top of the screen. The y coordinate is simple, it just needs to be zero. However, the x coordinate needs to be a random number between 0 and the SCREENWIDTH. To generate a random number we can use the Math object's GetRandomNumber(max number) method. However, we need to subtract the width of the ball from the SCREENWIDTH because if the x coordinate is SCREENWIDTH that would place the ball off the screen.


SCREENWIDTH
x, y coordinate of the ball is here. If x = SCREENWIDTH then the ball will be drawn off the screen.
By subtracting the width of the ball from the SCREENWIDTH, we can ensure that the ball is always drawn on the screen.

Finally line 27 moves the ball into its intial position before the game starts.

```
29 'Start screen
30 message = "Use the mouse to catch 10 balls."
31 GraphicsWindow.ShowMessage(message, "Catch!")
```

Line 30 assigns a string to the variable message which is then passed to the GraphicsWindow.ShowMessage() method. This method will display a message box on the screen with the message ("Use the mouse to catch 10 balls") and title "Catch!". The user will need to press 'OK' for the game to proceed.


Catch!

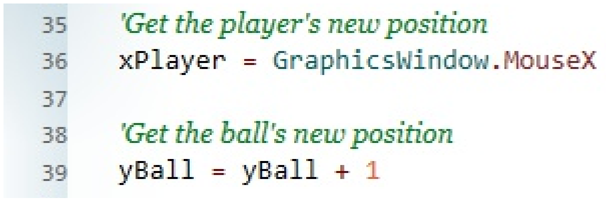
Use the mouse to catch 10 balls.

The rest of the program is inside an infinte While loop. In game programming this is often called the game loop. Inside the game loop all of the game logic, user interactions, collision detection and drawing updates are performed. If the code isn't well crafted and efficient then the game will run slowly and the user experience will be poor. Game designers usually aim for something like 30 - 60 frames per second.

```
33 While "True"
```

The rest of the program, which calculates the new positions of the ball and player; checks for collisions and a win; and finally redraws everything in its new position, is placed inside the While loop (lines 33 to 88).

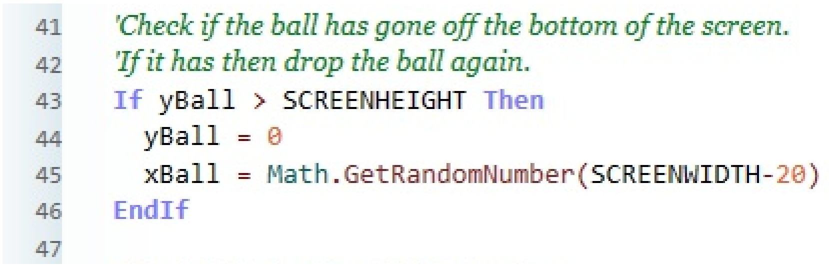
```
88 EndWhile
```



```
35  'Get the player's new position
36  xPlayer = GraphicsWindow.MouseX
37
38  'Get the ball's new position
39  yBall = yBall + 1
```

Once the program enters the While loop the first thing we do is capture the new position of the mouse with GraphicsWindow.MouseX and assign this value to the variable xPlayer. Remember xPlayer holds the x coordinate of the player's blue box. We don't need to modify the y coordinate of the player since that remains constant, it never changes from just above the bottom of the screen.

In line 39 we add 1 to the y position of the ball (remember it is intially at the top of the screen) which moves it down the screen. The ball will fall at a speed of 1 pixel every time the while loop is repeated. So if the while loop is repeated 60 times per second the ball will move 60 pixels per second.



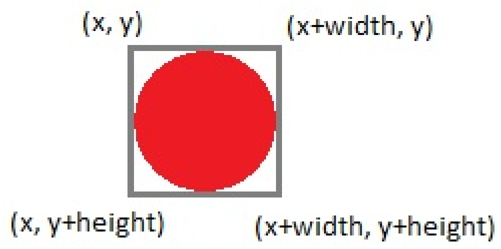
```
41  'Check if the ball has gone off the bottom of the screen.
42  'If it has then drop the ball again.
43  If yBall > SCREENHEIGHT Then
44    yBall = 0
45    xBall = Math.GetRandomNumber(SCREENWIDTH-20)
46  EndIf
47
```

After updating the position of the player and the ball we need to check to see if the ball has gone off the bottom of the screen. We do that with an If statement which will check whether the y coordinate of the ball is larger than the screen height. If that is true then we will need to reposition the ball at the top of the screen (line 44) and pick a random position along the width of the screen (line 45).

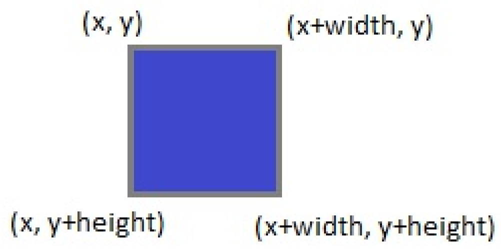
Collision detection

Collision detection is a very important aspect of game programming, without it we can not tell whether bullets have hit their targets or whether a player has landed on a platform. In this game we will use a simple technique of checking for overlapping bounding boxes.

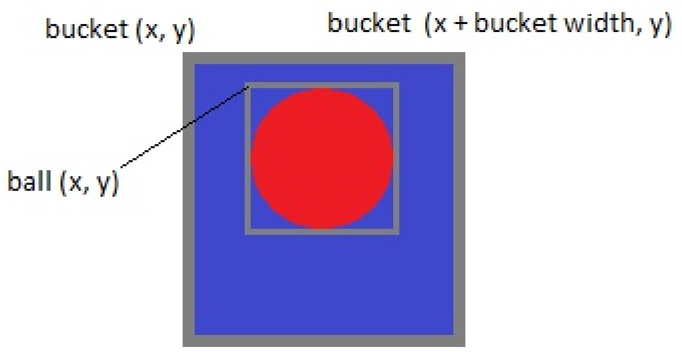
First we imagine an invisible box around the ball, the coordinate of the box is (x, y) the same as the ball. The far right top coordinate is (x+width of the ball, y), the bottom left of the box is (x, y+height of the ball) and the bottom right corner coordinate is (x+width of the ball, y + height of the ball).



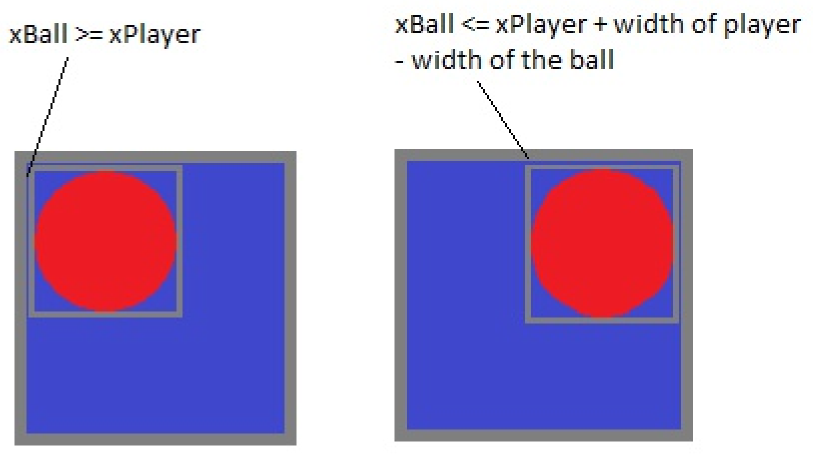
We also apply the same technique for the player's box as shown below.



Our collision detection is going to be as simple as possible. We are only interested when the ball is actually inside the bucket as the following diagram shows.

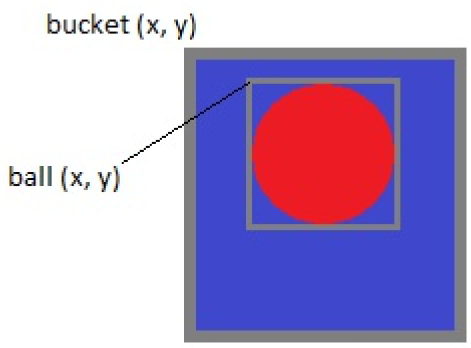


From the diagram we can see that the ball is only inside the bucket when its bounding box is completely inside the player's bucket bounding box. Mathematically this is only true when the x coordinate of the ball is greater or equal to the x coordinate of the box; AND in addition the x coordinate of the ball's bounding box must be less than or equal to the x coordinate of the player's box plus its width minus the width of the ball. See the diagram below.



```
48    'Check if the ball has hit the  bucket.
49    If (xBall>=xPlayer)And(xBall<=xPlayer+50-20)Then
50      If yBall >= yPlayer Then
```

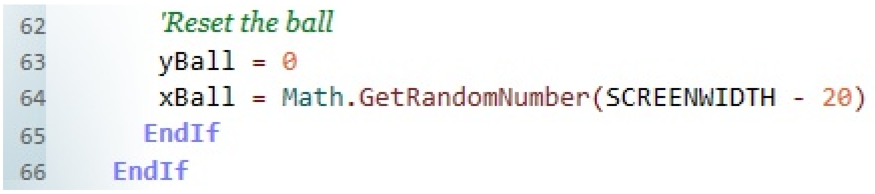
Line 49 checks that both conditions are simultaneously true using the AND operator. Then if the ball's x ordinate satisfies both of the conditions then we need to check whether the y ordinate is within the correct range (line 50). Specifically we need to check whether the y-coordinate of the ball is greater than the y-coordinate of the bucket (remember that y increases down the screen). See the diagram below.



If the ball has been caught in the bucket then several things need to be done. The variable score needs to be increased by one, the old scored needs to be erased from the screen and we need to play a congratulatory sound. Lines 52 to 60 accomplish this.

```
51
52        'Increase the score by one
53        score = score + 1
54
55        'Erase the old score by drawing a white rectangle.
56        GraphicsWindow.BrushColor = "white"
57        GraphicsWindow.FillRectangle(10, 10, 50, 20)
58
59        'Play a sound
60        Sound.PlayClick()
```

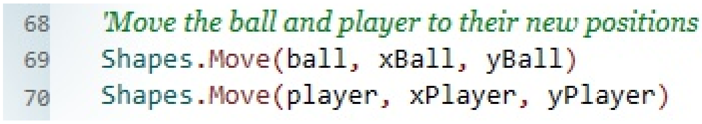
Finally we need to reset to position of the ball to a random position along the top of the screen - lines 62 to 64. The y-coordinate of the ball is easily dealt with by reassigning the yPos variable to zero. The getRandomNumber() function is used to pick a random number between 0 and the screen width.



```
62        'Reset the ball
63        yBall = 0
64        xBall = Math.GetRandomNumber(SCREENWIDTH - 20)
65      EndIf
66    EndIf
```

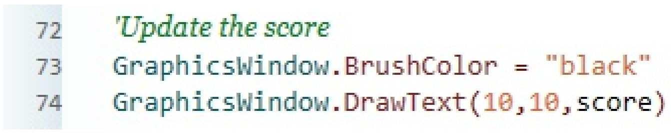
So ends the collision detection section of the code. The last section of code is executed whether or not there has been a collision between the bucket and ball. Lines 68 to 86 updates the position of the ball and the bucket, draws the score, checks whether the player has won and ends the game; and finally adds a short delay to adjust the speed of the game.

Lines 68 to 70 use the Move() function of the Shapes object to update and redraw the ball and bucket in their new positions.



```
68  'Move the ball and player to their new positions
69  Shapes.Move(ball, xBall, yBall)
70  Shapes.Move(player, xPlayer, yPlayer)
```

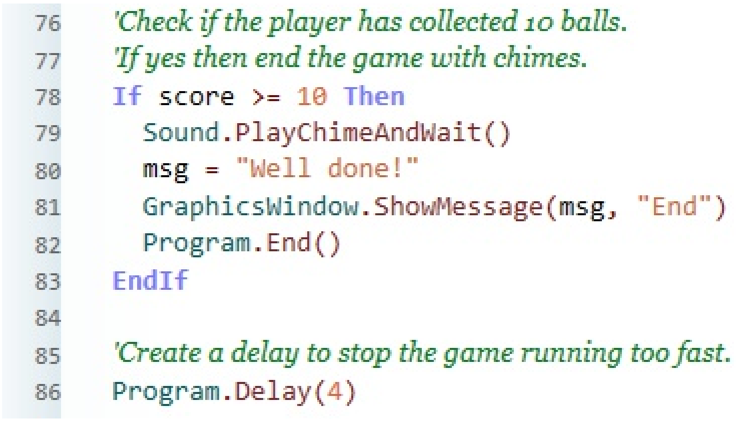
By setting the BrushColor to 'black' and using the GraphicsWindow's DrawText() function we are able to draw the score, which may have been updated if their was a collision.



```
72  'Update the score
73  GraphicsWindow.BrushColor = "black"
74  GraphicsWindow.DrawText(10,10,score)
```

Next, lines 76 to 83 check whether the player has caught ten balls. If the the check proves TRUE then a congratulatory sound is played, a message box saying 'Well done!' is displayed and then the program ends.

However, if the check proves FALSE then a small delay of 4 milliseconds is applied before the game loop is repeated again. The delay is used to make the game run at a suitable speed. The number 4 was chosen by trial and error.



```
76    'Check if the player has collected 10 balls.
77    'If yes then end the game with chimes.
78    If score >= 10 Then
79      Sound.PlayChimeAndWait()
80      msg = "Well done!"
81      GraphicsWindow.ShowMessage(msg, "End")
82      Program.End()
83    EndIf
84
85    'Create a delay to stop the game running too fast.
86    Program.Delay(4)
```

The program ends on line 88 with the Endwhile statement which will return the order of execution back to the beginning of the game loop on line 33. The game loop will continue to repeat endlessly until the program ends after the player has caught 10 balls.

Press play or F5 to play the game and if unfortunately the game doesn't run then you'll need to carefully check your typing for bugs. Otherwise enjoy the game.

Conclusion

That is the end of this chapter about writing programs for computers. If you have followed and understood the concepts introduced in the previous pages then you will have more than enough for a working knowledge of primary school programming.

Remember that as well as teaching skills it is more worthwhile for the children to use programming to solve problems, create applications; and explore patterns and simulations.