## VBugs

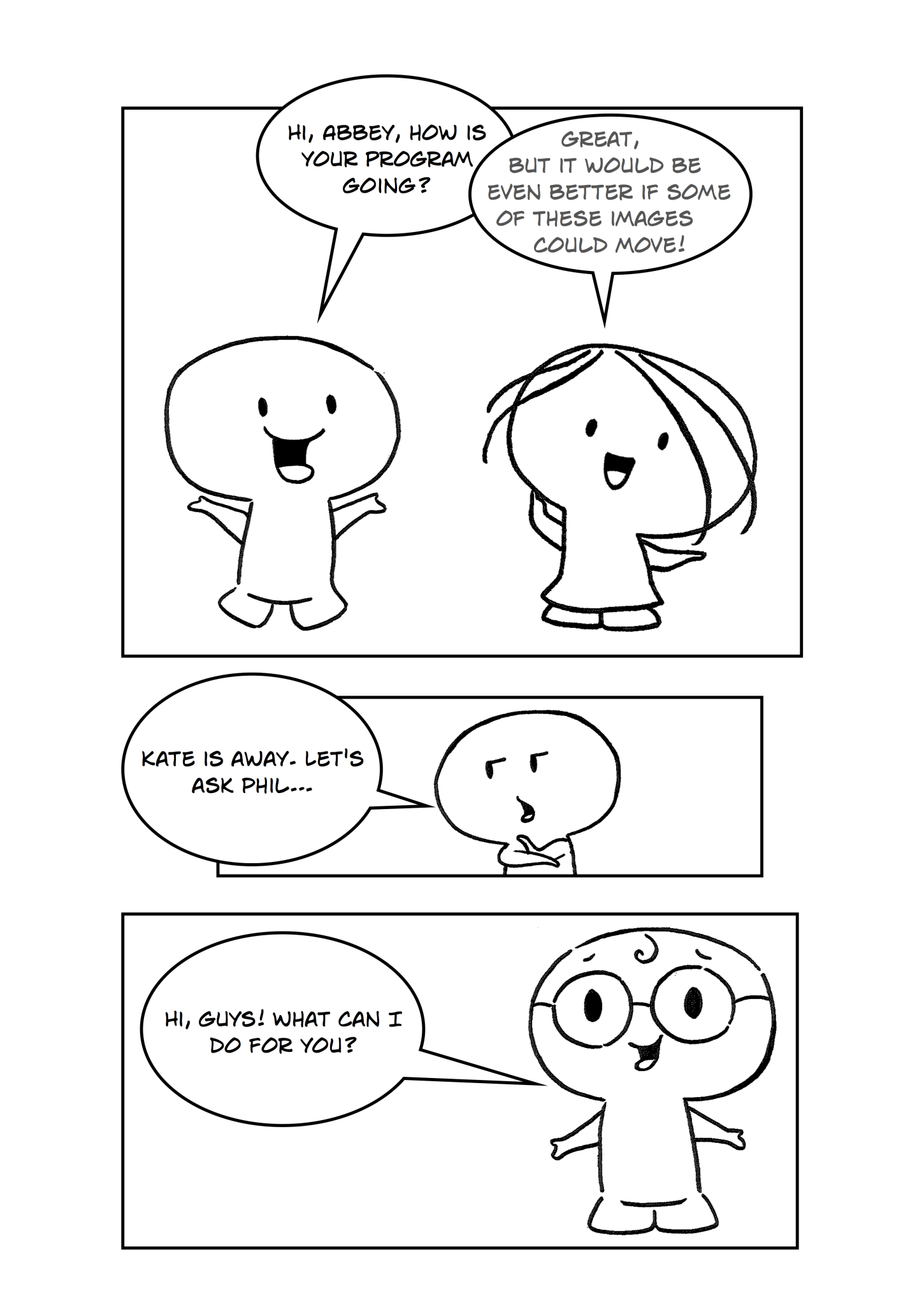
## Chapter 3

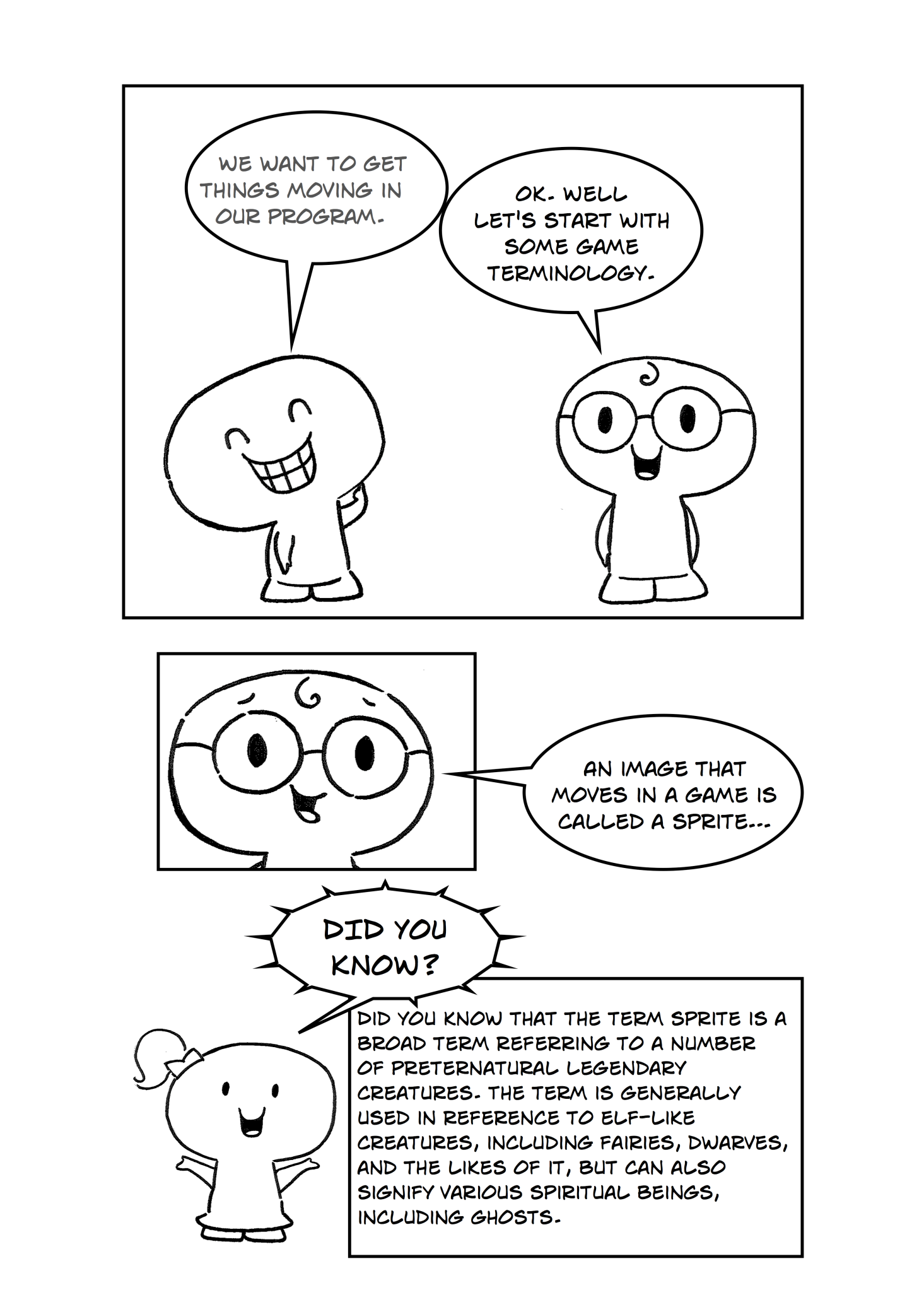
###### Movement

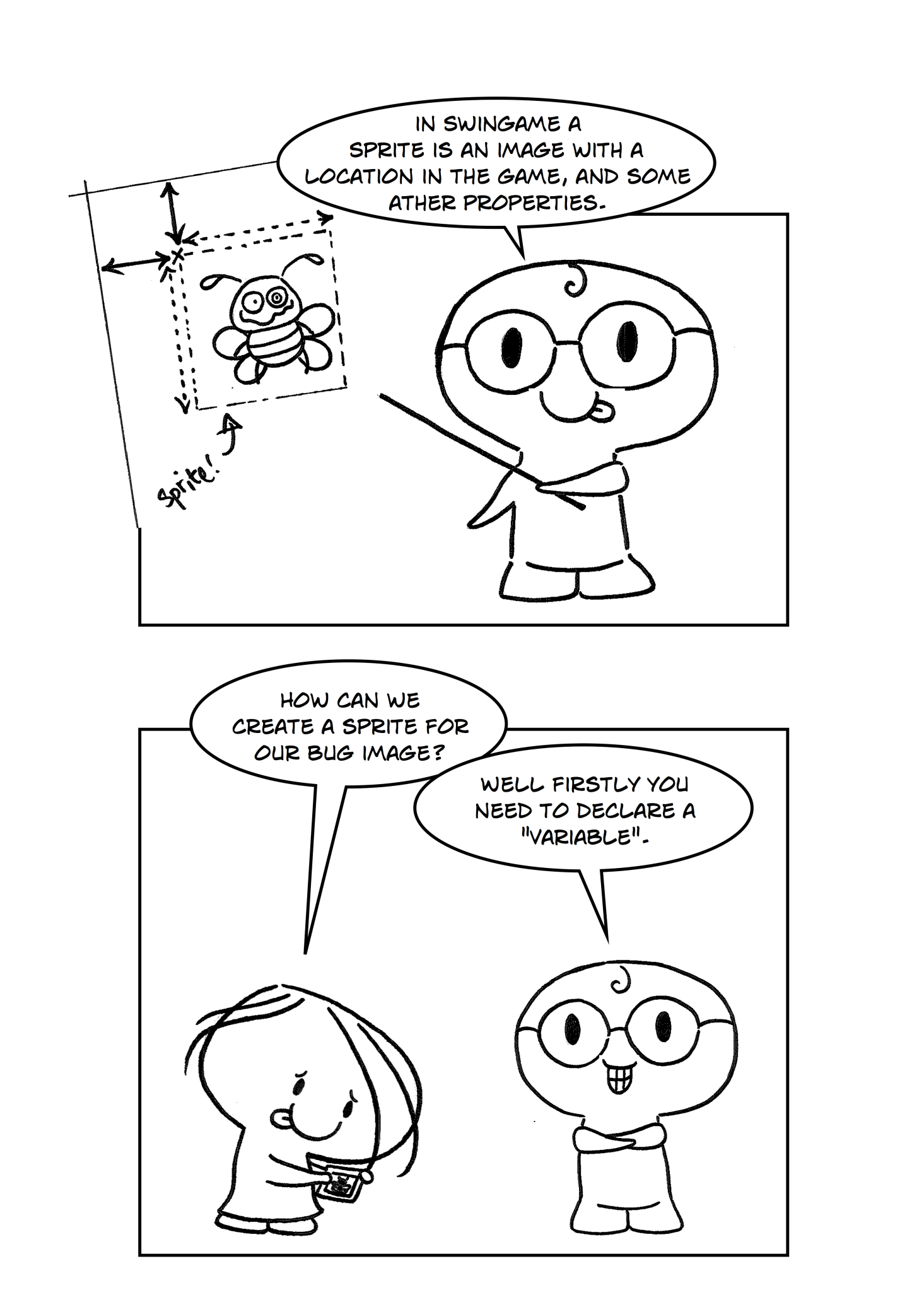


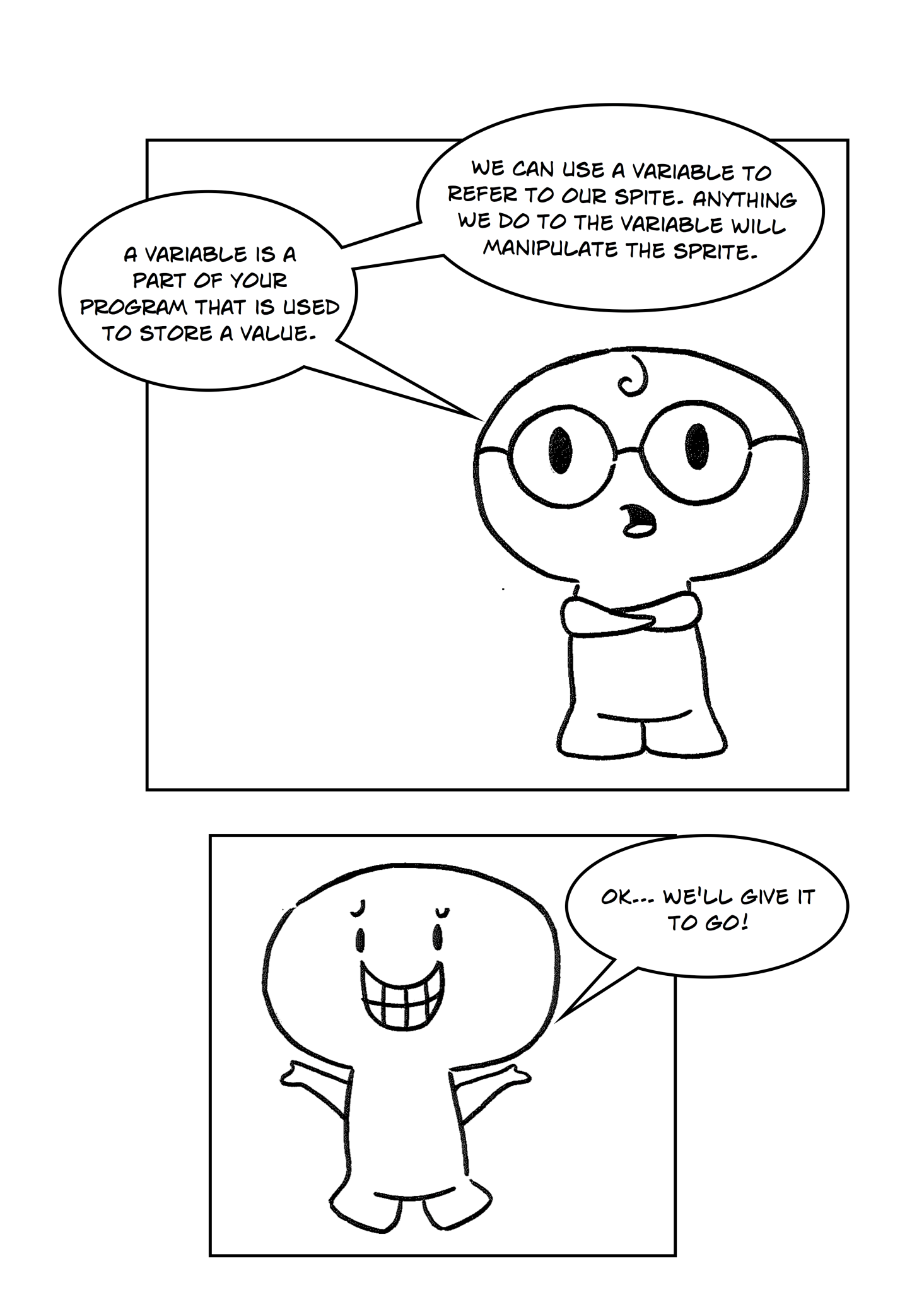
# Summary:

For this chapter you will be creating a new SwinGame project called “Bugs”. You will use your knowledge from the previous chapters and be introduced to some new terms and some game features such as movement of the main character.

****

****

****

****

## Part 1

### Sprites

A sprite is basically a small graphic (picture) that can be moved independently around the screen producing an animated effect. To create your spite you need to declare a variable which will refer to our sprite. This enables us to manipulate our sprite and make it to do what we want. Every time we create a sprite we need to free it at the end of our program so it will close down properly.

Open Visual Studio and create a new SwinGame project called “Bugs”. In “Game Logic.vb” delete everything between:

SwinGame.Graphics.ClearScreen()

and….

'Refreshes the Screen and Processes Input Events

-as you did in Chapter 2. Load image called “sprite.png” into your program as you did in previous chapter. Name the new image “sprite”: NewImage("sprite", "sprite.png") in “Game Resources.vb”.

cha 2 - worksheet.png*Question 1: What is the difference between a sprite and an image?*

Exercise 1: *Creating a sprite*

Make the following changes in your program and write your solutions to the worksheet:

1. Declare the variable “bug” which is a Sprite. To do so, put the following code after the LoadResources() where bug is the name of your variable and Sprite is the type of the variable:

|  |
| --- |
| 'Load Resources  LoadResources()  Dim bug As Sprite |

***Did you know:***

**** In VB.NET Dim stands for “Dimension” which. When you declare something you are setting aside some space in the computer’s memory for it to exist. There are many different types of data you can declare with a “Dim” statement such as Sprite, Integer (round number) boolean (Yes or No) and many more.

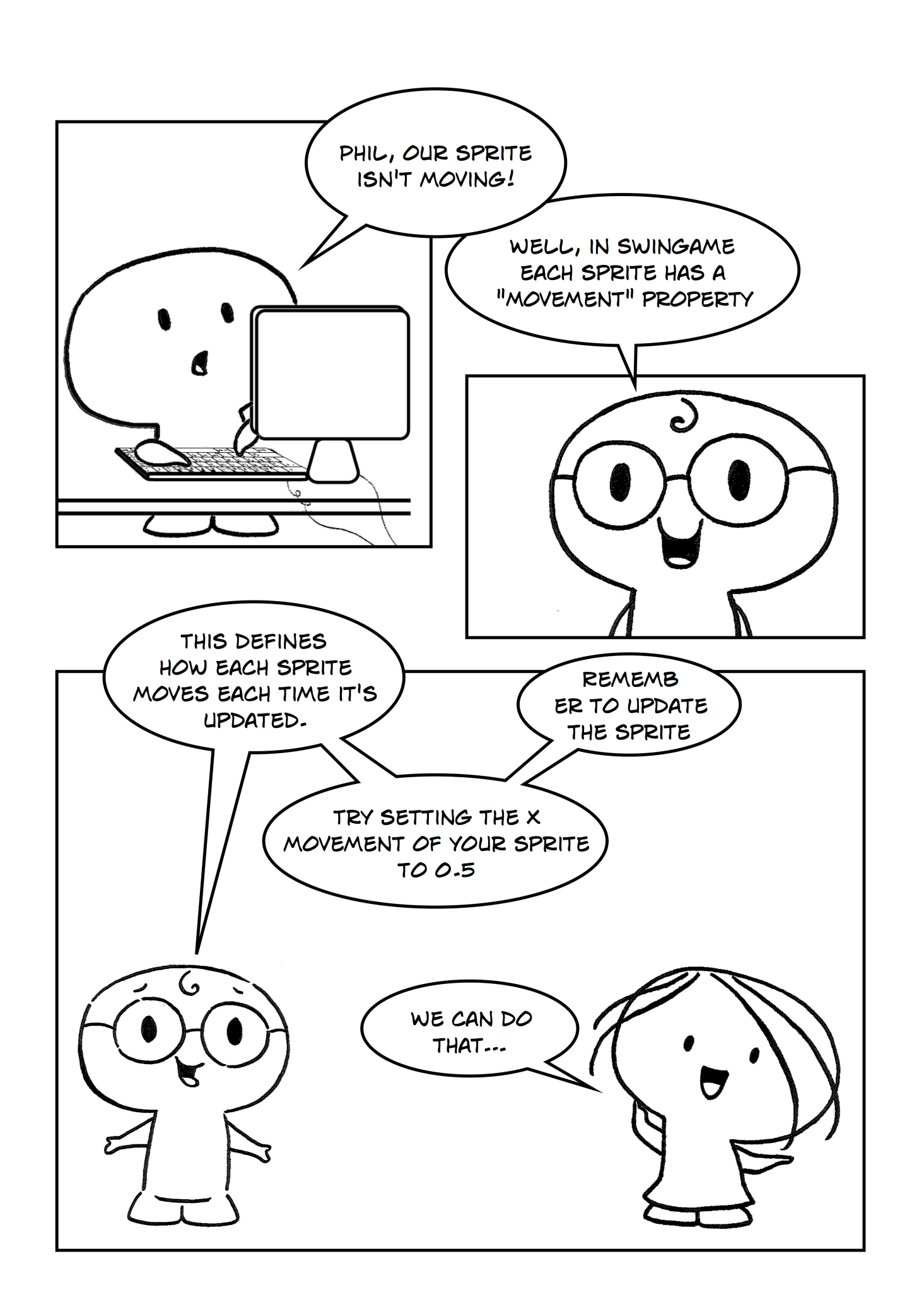
1. Now we can create a sprite in the program on the next line after the variable declaration use:

variableName = Graphics.CreateSprite(GameImage("nameOfImage")) after variable declaration.

1. Then draw the sprite on the screen use:

Graphics.DrawSprite(variableName)inside the Game Loop.

1. Put Graphics.FreeSprite(variableName at the end of your program, after FreeResources(), to free the sprite when the program closes.
2. Press the "StartDebugging" button at the top of the screen (looks like a green arrow arrow.jpg, F5 works too) to see what it does.

****

## Part 2

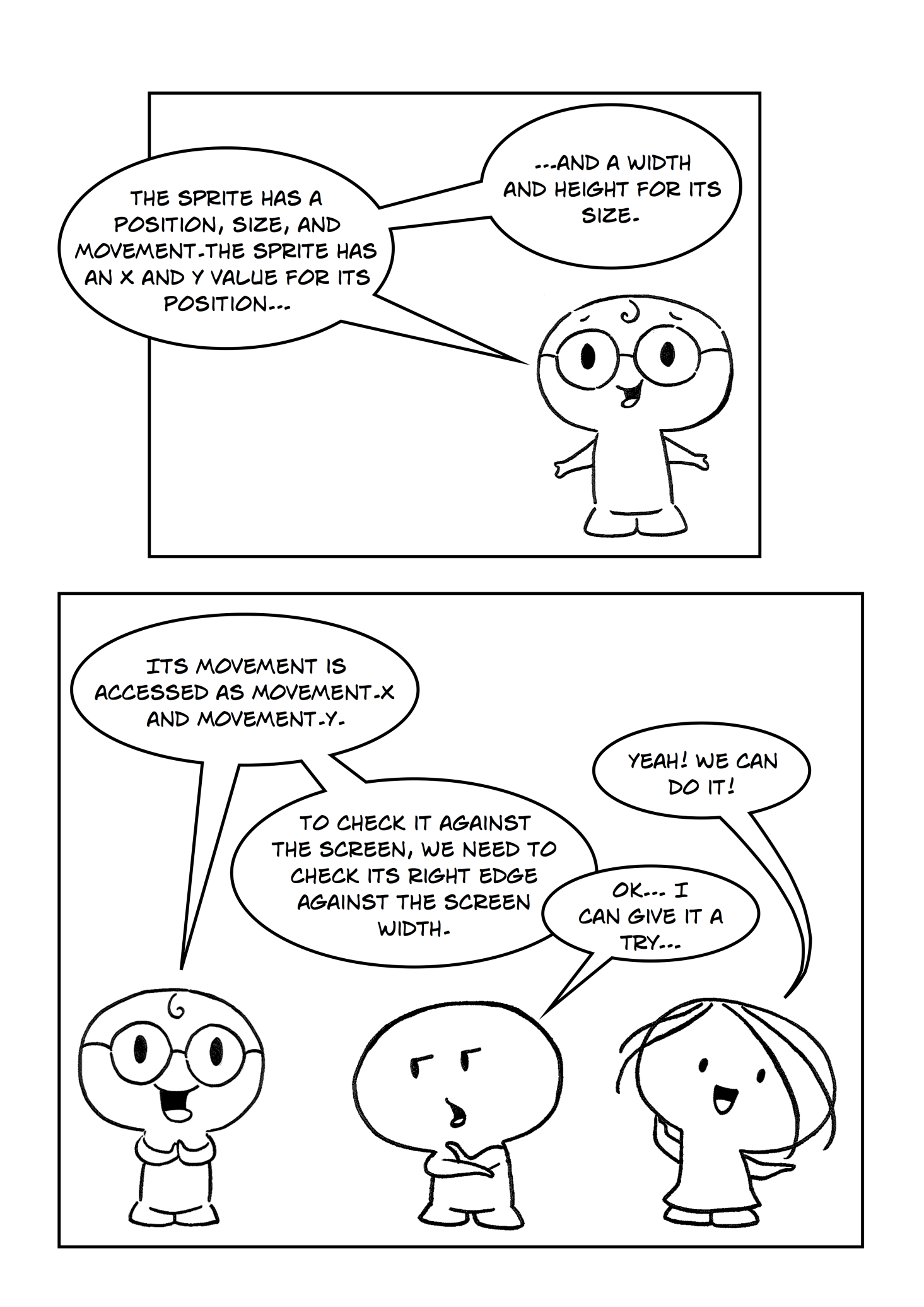
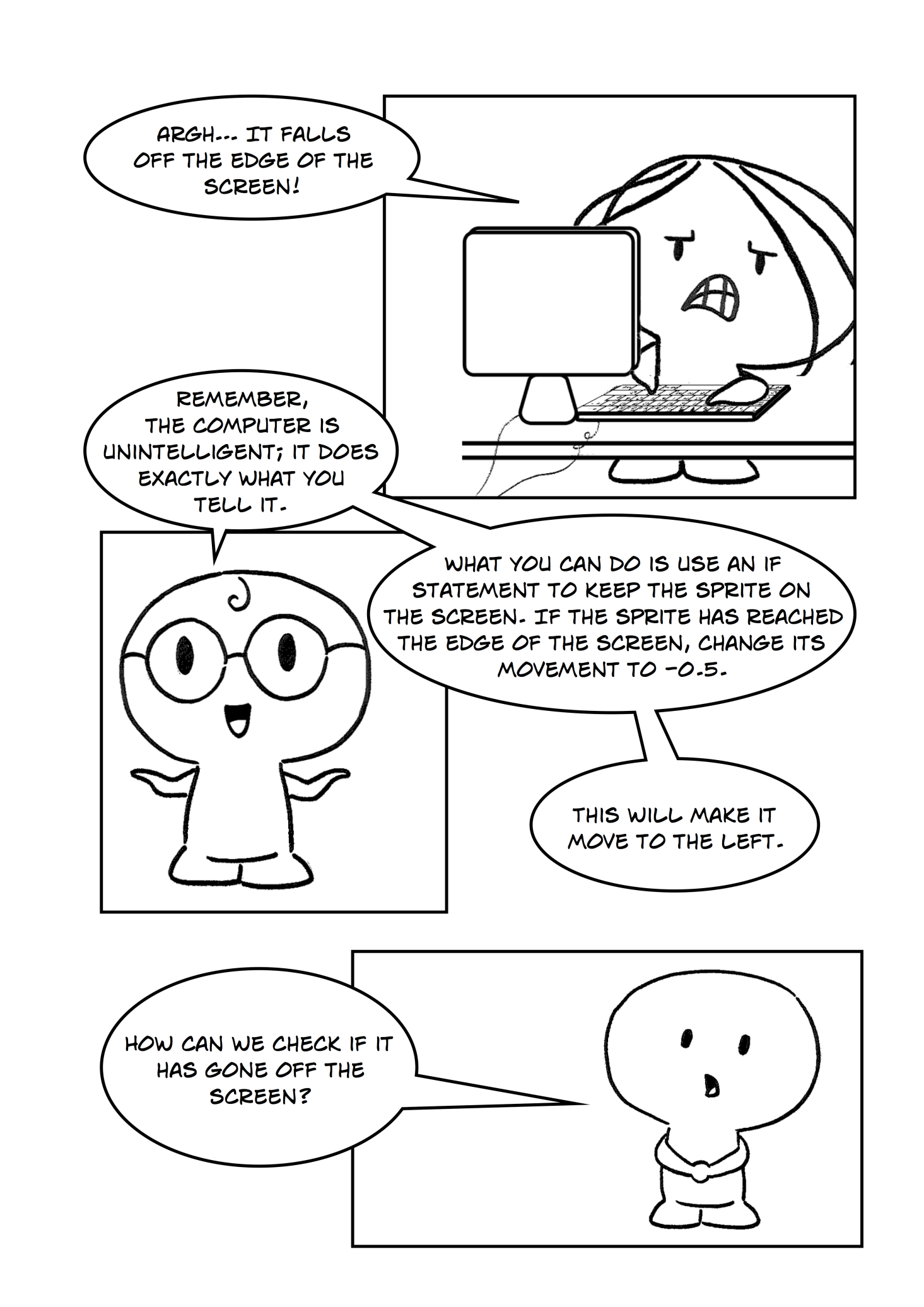
### Movement

The “Movement” property defines how much the sprite moves each time the screen is updated. The movement is defined by X and Y values. X is the amount of pixels that the sprite moves horizontally, and Y is amount of pixels that the sprite moves vertically. In order to see the exact movement of our sprite on the screen, we need to update our sprite inside the Game Loop.

Exercise 1: *Making the sprite to move*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Assign Movement.X of your sprite to 0.5. To do so, use variableName.Movement.X = 0.5, and put this code just before the start of the game loop.
2. In order to see how our sprite moves, it needs to be updated within the loop. Use Graphics.UpdateSprite(variableName) after we have drawn it to the screen in the code.
3. Press the "StartDebugging" button at the top of the screen (looks like a green arrow arrow.jpg, F5 works too) to see what it does.

****

## Part 3

### Keeping the Sprite On Screen

Each sprite has a position, size and movement. The position is defined by X and Y values of the sprite. The size is defined by width and height of the sprite. The movement is also determined by X and Y values which are the number of pixels the sprite is being moved each time the screen refreshes. The X and Y of movement can be accessed by Movement.X and Movement.Y (e.g. bug.Movement.X = 0.5).

In order to keep the sprite on the screen we need to use all parameters of the sprite and an “If” statement. The “If” conditional expression is one of the most useful control structures, it allows us to execute an expression “if” a condition is true. The syntax looks like this:

|  |
| --- |
| If condition Then [statements] End If |

If the condition is true, the statements following the Then [keyword](http://www.startvbdotnet.com/language/ifthen.aspx) will be executed. If they are no then it will skip past the End If and not perform those statements. If statement could also contain the Else expression:

|  |
| --- |
| If condition Then [statements] Else If condition Then [statements] - - Else [statements] End If |

That basically means that if the condition is true, the statements following the Then [keyword](http://www.startvbdotnet.com/language/ifthen.aspx) will be executed, else, the condition following the Else If will be checked and if true, the second block of statements will be executed, else, the statements in the Else part will be executed.

For example; milk can’t go below 1 or above 4 degrees Celsius. Let’s say we wanted an alert if the temperature in a fridge went beyond those limits we could write it in Pseudocode in the following way:

|  |
| --- |
| If temperature > 0 and temperature <5 Then  Write “OK” Else  Write “Alert” End If |

***Did you know:***

**** Pseudocode is used to describe code to people. It has the same structure and logic as programming code but we can leave out things that only the real code needs such as variable declarations or exact syntax (rules, spelling and order). It makes it easier for people to understand what is actually going on in the code.

cha 2 - worksheet.png*Question 1: Write the pseudocode for an “IF Statement” where if x is 200 it says “Perfect score” otherwise it divides x by 200 and displays the result to the screen.*

cha 2 - worksheet.png*Question 2: Write the pseudocode for an “IF Statement” where if x is above 40 it says “Very Hot” if it is above 30 it says “Hot” otherwise it says “Mild”.*

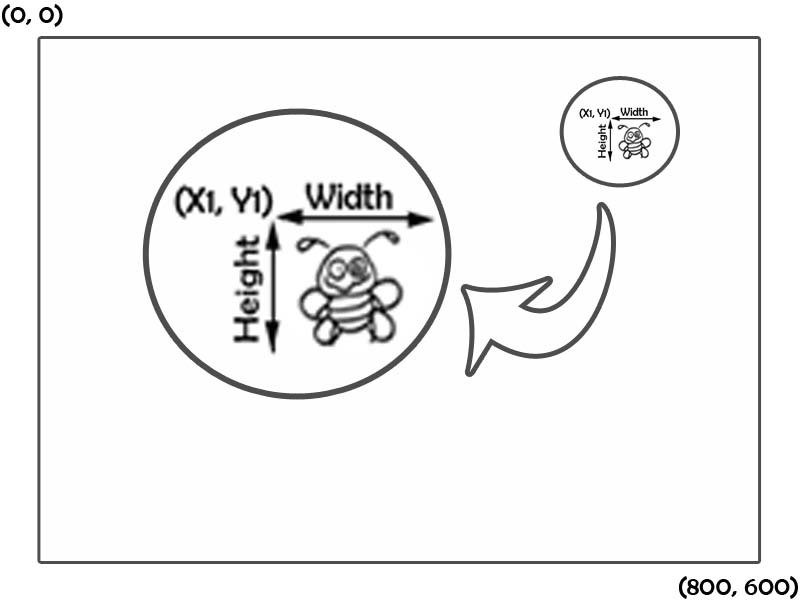
In order to check whether our bug falls off the edge we need to use a simple If statement. To do so, we need to check its right edge against the screen width. The right edge is the current position of the sprite plus the width of the sprite (remembering that the position is taken from the top right corner), as shown in Figure 1:

Figure 1

The logic for checking whether the sprite is within the screen is shown in the NS diagram in Figure 2:

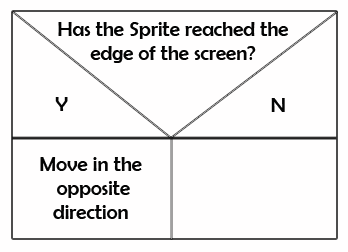


Figure 2

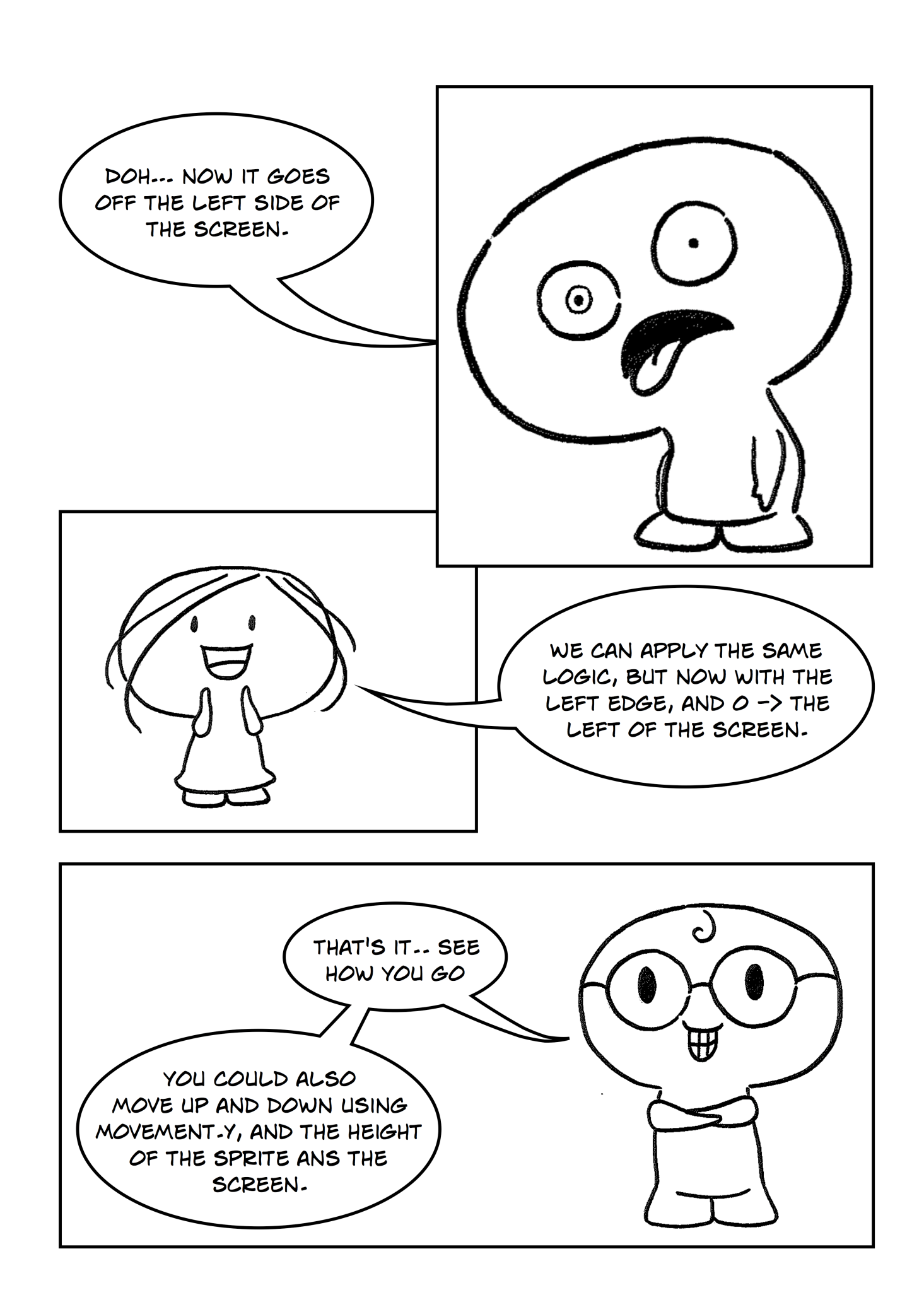
Exercise 1: *Stopping the Sprite from moving off the right edge of the screen.*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Add the following code to the Game Loop after the line containing Graphics.UpdateSprite(bug) then on your worksheet describe what their purpose is:

|  |
| --- |
| If variableName.X + variableName.Width >= Core.ScreenWidth Then  variableName.Movement.X = -0.5  End If |

1. Press the "StartDebugging" button at the top of the screen (looks like a green arrow arrow.jpg, F5 works too) to test the code then close the window.

****

## Part 4

In order to stop the Sprite from moving off the left edge of the screen we need to use the same logic as we used previously. The difference here is that unlike the right edge which is determined by the position of the Sprite plus Sprite’s width, the left side of the Sprite is defined only by its position (remembering that the position is taken from the top left of the object).

Exercise 1: *Stopping the Sprite from moving off the left edge of the screen.*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. *Add the code to the Game Loop that will test if the bug’s X position is less than or equal to 0 and if it is set its movement to positive 0.5. Place this code after the code from the previous exercise:*
2. *Press the "StartDebugging" button at the top of the screen (looks like a green arrow arrow.jpg, F5 works too) to test the code then close the window.*

So far the Sprite is moving horizontally; by assigning a value to Movement.Y you can move the Sprite in all different directions.

Exercise 2: *Changing the movement direction.*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. *Assign Movement.Y of the Sprite to 0.5, this can be done in the same way as shown in part 1 > exercise 1.*
2. *Write the code which will stop the Sprite from moving off the top edge of the screen; use the same logic as in part 3 & 4 exercises.*
3. *Write the code which will stop the Sprite from moving off the bottom edge of the screen; use the same logic as in part 3 & 4 exercises(hint: you will have to think about Height in this case instead of Width):*

cha 2 - worksheet.png*Question 1: Understanding Direction*

1. *A bug travelling with and X movement of 2 and a Y Movement of 2 would go in which direction.*
2. *A bug travelling with and X movement of 2 and a Y Movement of 2 would go in which direction.*
3. *A bug travelling with and X movement of 2 and a Y Movement of 2 would go in which direction.*
4. *A bug travelling with and X movement of 2 and a Y Movement of 2 would go in which direction.*

## Part 5:

### A Second Bug

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

Exercise 1: *Two Bugs*

Add a second Sprite to your program, follow the same steps you did for the first bug to achieve this. You will need to give your second bug a different name in the code (e.g. bug1).

saveicon.png Remember to save your project (File – Save All). Once you have finished then you can close Visual Studio or move on to the next chapter.