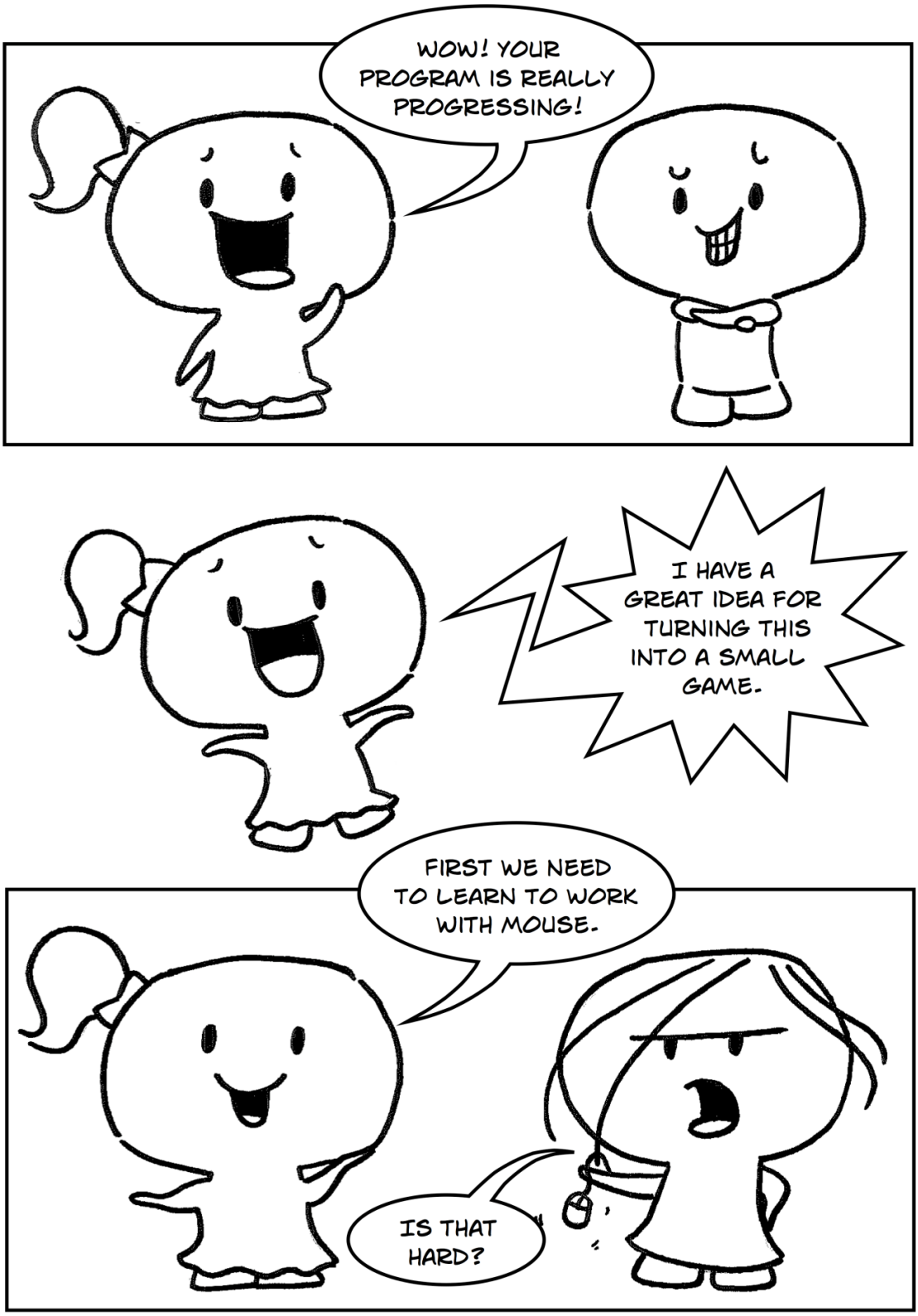
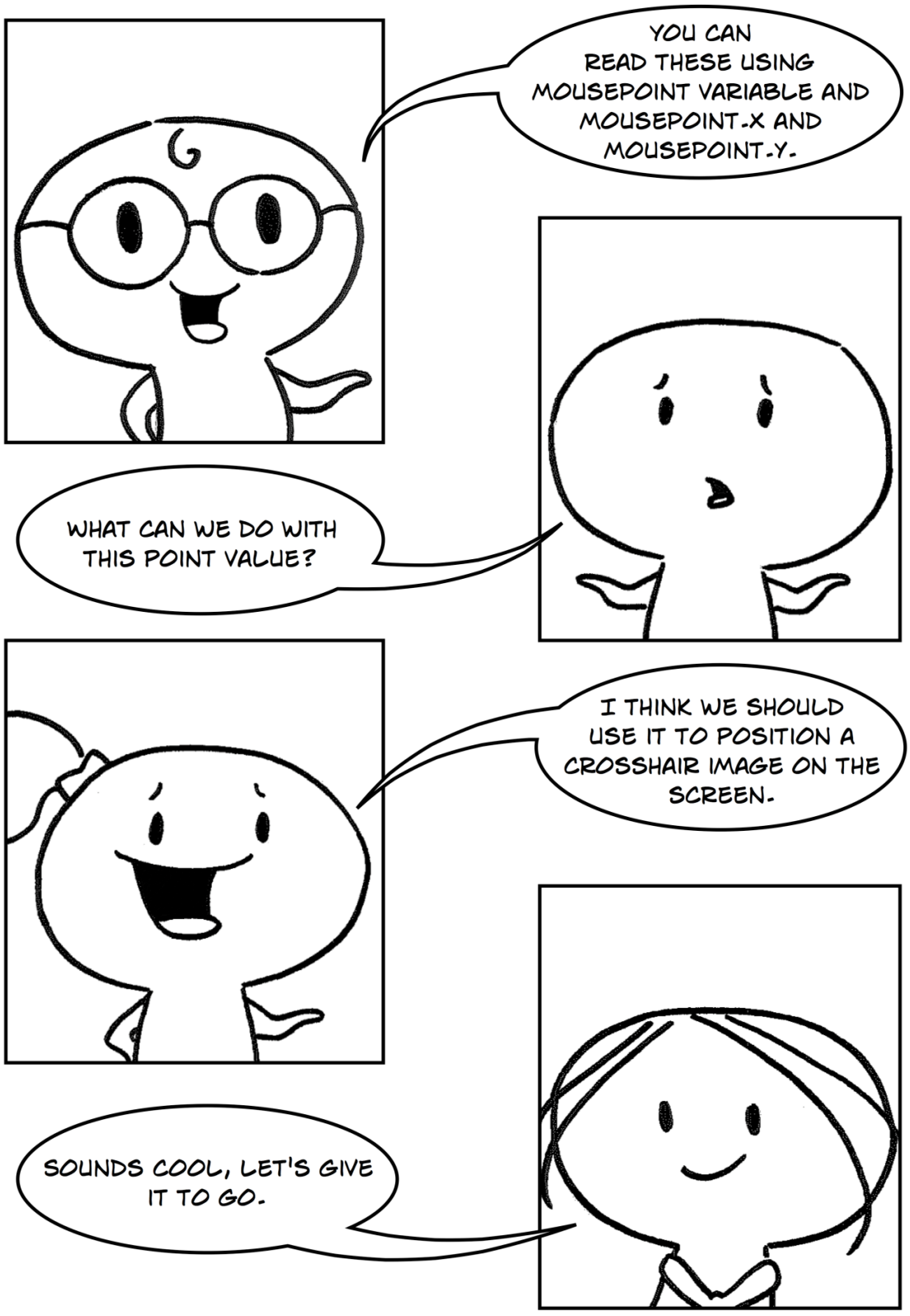
Chapter 5

"Mouse and Animation"

Summary:

In this chapter you will need to modify the solution from the previous chapter. You will learn how to replace default mouse point to a custom picture, how to handle mouse input and use animated sprites. All resources will be provided for this chapter.





Part 1

In SwinGame you can reach the position of mouse through the variable of Point2D type which has to be declared outside the Game Loop. For example the variable name is mousePoint, so you can reach its X position through mousePoint.X and Y though mousePoint.Y.

Based on knowledge about the mouse onto the SwinGame screen, you can replace the normal mouse point with a crosshair image. To do so, inside the Game Loop, because the program should continuously redraw the image that you want, you have to hide the current mouse point and use a custom image instead.

To hide the current mouse pointer use `Input.ShowMouse(False)` (this should be made outside the Game Loop to increase the speed of your program). This is build-in function that takes True parameter by default. Now you can place a custom image (crosshair) instead of the original one. To do so, we need to know the position of the mouse on the screen. We can get mouse position at a time by assigning a variable mousePoint to `Input.GetMousePosition()` inside the Game Loop. Then we can draw the crosshair image at the position of the mouse with `Graphics.DrawBitmapOnScreen(GameImage("ImageName"), mousePoint.X, mousePoint.Y)` function.

Exercise 1: *Replacing the original mouse point to the custom one*


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

1. Inside the Game Loop tell the program to draw target.png instead of the original mouse point.

NOTE: You should load target.png into your program, which has size 40x40 pixels. Mouse position is a position of the left top corner of the mouse on the screen. In order to put the target in the right way (the shoot point is middle of crosshair image), you should consider the size of the image. So, the middle of the image will have a position of X = `mousePoint.X - 20` and Y = `mousePoint.Y - 20` as shown in Figure 1.

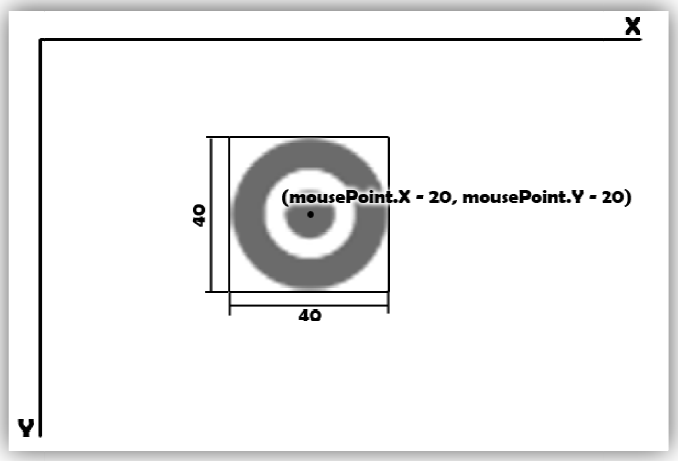


Figure 1

Part 2

In order to play sound when clicking on a bug, you need to follow the logic represented in Figure 2.

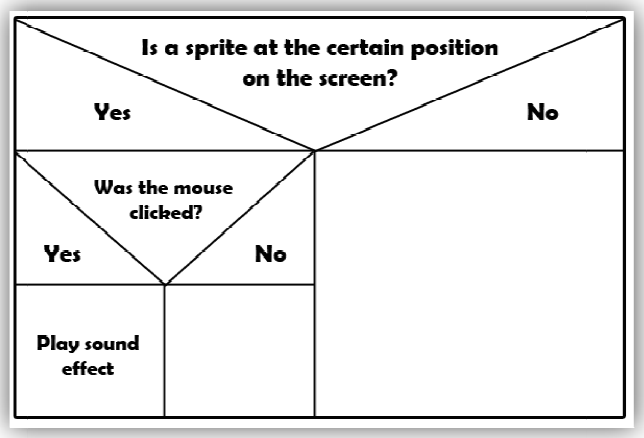


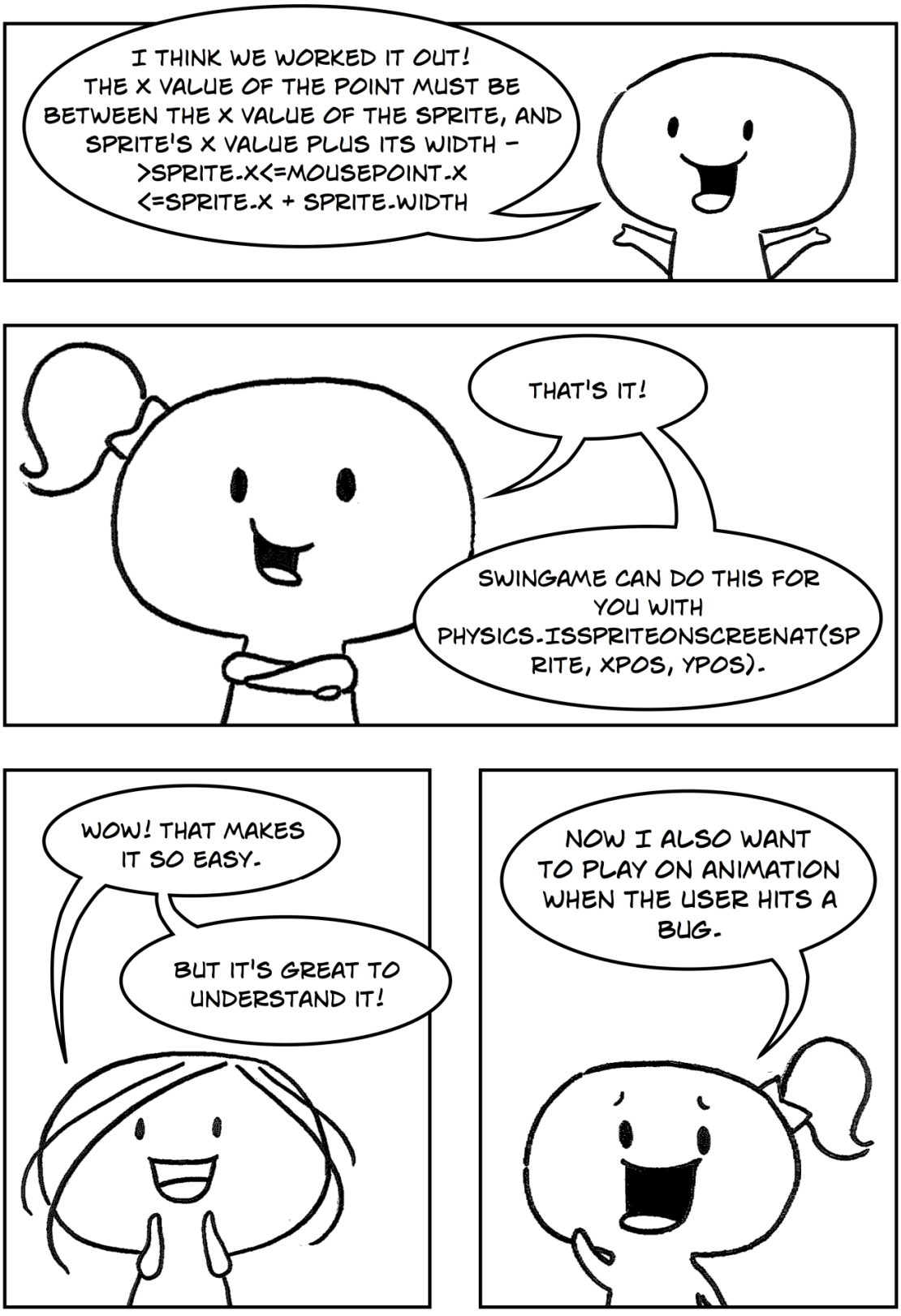
Figure 2

To check whether a sprite at a certain position on the screen, you should use the `Physics.IsSpriteOnScreenAt(bug, mousePoint.X, mousePoint.Y)`, where bug – is your sprite variable, mousePoint.X and mousePoint.Y are coordinates of the mouse on the screen. To check whether the mouse button was clicked you need to use `Input.MouseWasClicked(MouseButton.LeftButton)`. Both of functions are buil-in functions which return True or False values. Because the mouse is moving we need to check its position at a time, so we need to write our code inside the Game Loop.

Exercise 1: *Playing sound when clicked on a bug*

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

1. Write the code that will enable to play sound effect when a bug was clicked. Use knowledge from previous chapters and follow the logic above.



Part 3

SwinGame supports animated sprites. Basically animation in SwinGame is a sheet with a number of images which are divided into "cells". Each cell has same width and height. Each cell is shown a number of times, before moving to the next one, as higher the number of times to show each cell as slower the animation. You have to load a sheet of images into your program as a normal image. Figure 3 represents a sheet of images that we will use as animation in our program.

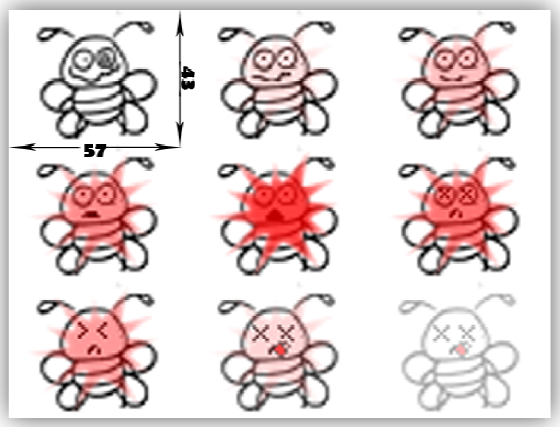


Figure 3

In our case each cell has 57 pixels width and 43 pixels height. To create an animation firstly we have to declare the variable which will represent our animated sprite. It should be done outside the Game Loop. You have to declare a variable as a Sprite and assign it to the following function: `Graphics.CreateSprite(GameImage("deadBug"), 20, 10, 57, 43)`, where 20 – is the number of times each cell will be showen, 10 – total number of cells, 57 – width of each frame and 43 – height of each frame. Note that the total number of cells is bigger than the number of cells in the picture; this allows you to hide the animation after it was played. If you will leave the exact 9 frames as total number of frames, the last image in the sheet will stay on the screen.

Exercise 1: *Declaring the animated sprite variable*

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

1. Declare a new animated sprite variable with a variable name as "deadBug". Use CreateSprite() function to assign to the variable.

NOTE: Do not forget to load a sheet of images into your program before using it. If you have two or more bugs, do the same procedure for all of them.

When a bug is clicked the animation of dying bug should be played. You should fix the position where the bug was killed and play animation at exactly this position. To do so, inside the if statement which checks whether the bug was clicked and, if it is so, plays a sound, you need to assign bug movement to 0, so it will not move further:

```
bug.Movement.X = 0
bug.Movement.Y = 0.
```

Then you should replace the current bug with an animation sprite at exact position when the bug was clicked. To do so, firstly you have to assign the position of the animation to the position of the bug:

```
deadBug.X = bug.X
deadBug.Y = bug.Y.
```

This will enable you to play an animation at the position when the bug was clicked. But at this point, if you run you program, you will see that the bug will just stop at the position it was clicked. To run the animation, you have to replace the bug with the animation of dying bug by assigning bug variable to animation variable: `bug = deadBug1`. If you run you program again and click on the bug, you will see that the animation is repeating to play. To prevent animation to play, you have to type this code:

`deadBug.EndingAction = SpriteEndingAction.Stop`, which stops playing animation after the first time.

The logic for playing animation action is shown in Figure 4.

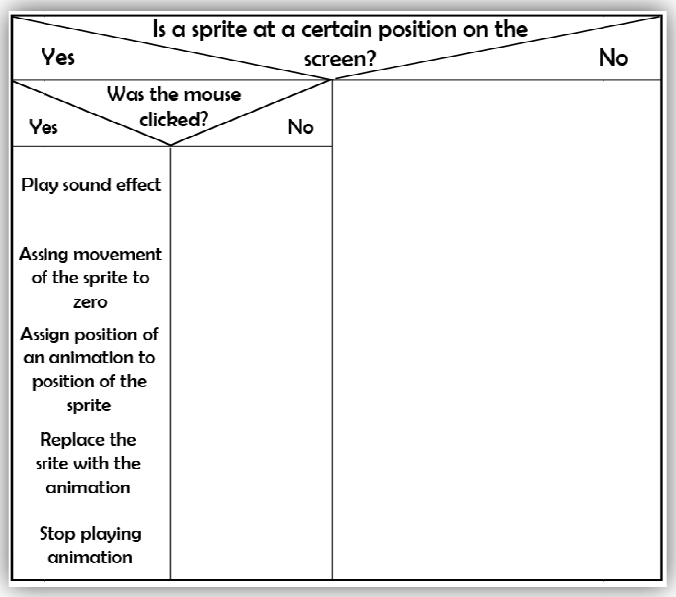


Figure 4

Exercise 2: *Playing animation when a bug was clicked.*

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

1. Write the code that plays animation when a bug was clicked. Use information above to complete this task.