Smalltalk Programming
Lesson 15

In the last lesson you coded the ability for your ship to shoot. After testing your code changes, you noticed that even though the shot displays, it does not move. A shot from your ship will not be much good if it does not move towards a target! Today, you will make the necessary code changes in order for your shot to move.

1. Type the method below in your `Shot` class, which is used to move your `Shot` instances.

move

```
self position: (self position) - (0@5).
self top < owner top ifTrue: [self delete]
```

2. Look at each line of code. Can you figure out what each line of code is doing?

3. Now that your code has been added, did your `Shot` instance(s) move?

4. Your `Shot` instance(s) did not move yet because the Morphic animation system has not been enabled for your `Shot` morph.

5. Morphic provides the ability to animate Morphs. This animation is handled by using `step` and `stepTime` on Morph instances.

1. The `step` method works kind of like having a heartbeat. Just as a heartbeat means you are alive and keeps you moving, when the `step` method is declared, Morphic will animate the Morph using the code within the method. It will run this code at regular intervals.
2. The `stepTime` method works kind of like how fast a heart is beating. This sets the interval for your `step` method. The smaller the number is, the faster your step interval will be. The larger the number is, the

slower your step interval will be. This number says how much time you want to wait before the `step` method runs again.

6. Before you make your Morphic animation code changes, now would be a good time to explore and interact with your Morphs. This exploration will help you to better understand how stepping works with Morphs.

7. Open an object explorer on your shooter game. If you do not remember how, press the middle mouse button while the mouse cursor is over your game screen. The Morphic halo will display. Select the wrench icon on the right side and then select “explore morph”.

8. Select and expand “submorphs”. Then select and expand a `Shot` instance. Your explorer window should look similar to Figure 1 below.

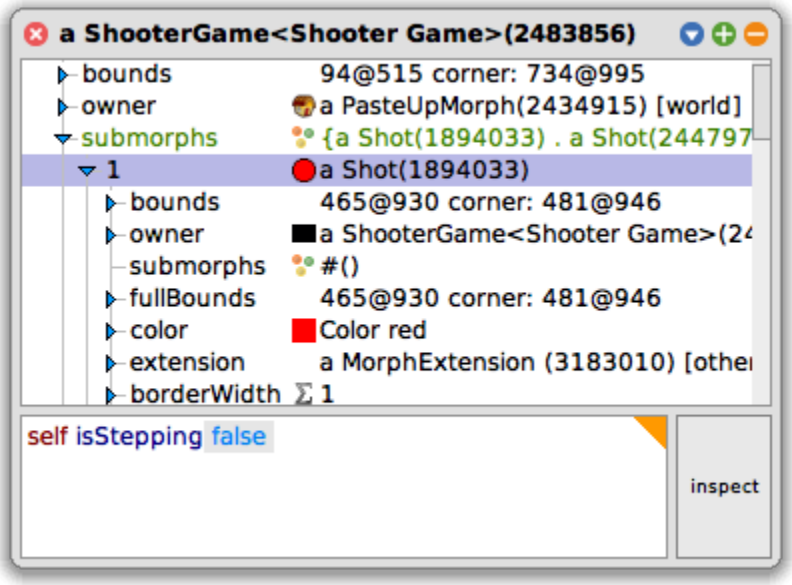

Figure 1: Object explorer on a Shot instance

9. Type the following expressions in your explorer workspace window, then select “print it” (or Ctrl-p) for each line. What is the result for each expression?

```
self wantsSteps.
self isStepping.
```

10. The results for your expressions were “false”. This shows that Morphic animation has not been enabled for your `Shot` instance(s). Keep your explorer window open for now.

11. Type the code below in your `Shot` class and then save the changes.

step

```
    self move
```

12. Did your `Shot` instance start moving?

13. Select “print it” (or Ctrl-p) for each of your 2 stepping expressions again. What are the results?

14. You will see that your `Shot` instance wants to be included in the Morphic animation system. However, the last expression shows that stepping is not yet enabled for your `Shot` instance.

15. Your `Shot` instance(s) are not moving because they were created *before* the Morphic animation system was enabled with the `step` method. Morphic instances created before creating the `step` method will default to not step until they are told to do so.

16. Enable stepping for the `Shot` instance that you have selected by selecting “do it” on the expression below.

```
self startStepping.
```

17. Your shot is now moving. If you have more shots displayed, you can move all of them by typing the following code in your object explorer workspace or in a Workspace window.

```
Shot allInstances do: [:shot | shot startStepping].
```

18. Now that the `step` method exists in your `Shot` class, all new shots will be included in the Morphic animation system. You will not need to do anything special to make this happen. The reason you needed to do something special this time is because these instances were created *before* the `step` method was added.

19. You might notice that your shots are not moving as fast as you might like them to. Type the following code to change that.

stepTime

```
    ^ 33
```

20. The default `stepTime` value is 1000. The larger the `stepTime` number is, the more time that will go by before the next `step`.

21. Test your new code. What do you notice? What else needs to be done?

22. Save and Quit your Smalltalk image.