# 7 Repetition: A Counted Loop

In chapter 6, we looked at If/Else statements for deciding whether or not an action is taken. In a way, we might think of If/Else statements as allowing Alice to make a choice about an action -- a form of control of program execution. In this chapter, we look at a second kind of control statement, one that loops. A looping statement is one that *iterates* or *repeats* actions over and over again.

We introduce looping statements with the *Loop* instruction. In Alice, a Loop is a block of one or more instructions that are run again and again a given number of times. We call the number of times the *count*. For this reason, the Loop may be referred to as a *counted loop*. The counted loop is very much like the "for loop" construct found in many other programming languages. But, Alice makes the loop easier to use by allowing the programmer to directly specify the count or use a question to determine the count.

Loops are the simplest form of repetition, and we will examine repetition in much greater detail over the next few chapters.

## 7-1 Loops

### Introducing repetition

As our worlds become more sophisticated, the code tends to become longer. We also encounter the need to repeat an animation instruction, or several animation instructions, to make the same action occur over and over. It is quite possible that you have already built worlds where you needed to drag the same method into the editor several times. If an animation was to be repeated, the only way (until this point) was to drag the method into the editor as many times as needed. In this section, a programming construct is introduced that allows instructions to be repeated. The focus of this section is on how repetition works, and how to write repetition code in Alice.

### The need for repetition

In the world shown in Figure 7-1-1, a baby bunny has snuck into his neighbor's garden. The bunny has spotted the nice broccoli shoots the gardener planted a few weeks ago -- broccoli is obviously on the bunny's menu today. Our task is to write a program to animate the bunny hopping over to munch a few bites of the broccoli. But, just as the bunny reaches the broccoli papa rabbit appears at the gateway to the garden and taps his foot in dismay. The bunny lowers his head sadly and hops a quick retreat out of the garden.

**Figure 7-1-1. Bunny in Garden**

In this description of the program, let's assume that we have already written a method for the bunny named *hop*. The hop method enables the bunny to hop forward by moving his legs up and down at the same time as the entire bunny moves forward. A possible implementation of the *hop* method appears in Figure 7-1-2. (Of course, sound is optional.)


**Figure 7-1-2. *Bunny.hop***

Let us suppose that the bunny is eight hops away from the broccoli. One possible way to animate eight bunny hops is given in Figure 7-1-3. A *Do in order* block is placed in the *World.my first method*. Then, the bunny points at the broccoli, and the *Bunny.hop* instruction is dragged into the editor eight times, as illustrated in Figure 7-1-3.

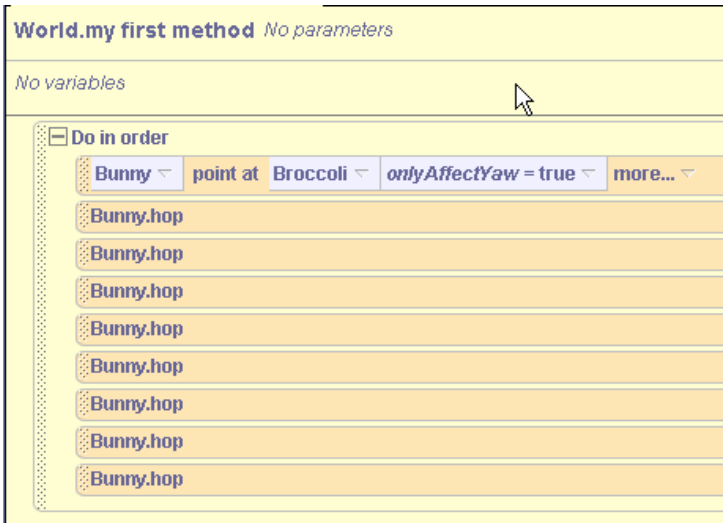



**Figure 7-1-3. Eight hops**

Actually, there is nothing wrong with this code. But, it was tedious to drag eight *Bunny.hop* instructions into the World script. Alice provides a special program construct, called a **Loop**, to allow repeated motion without having to do so much work. A Loop is commonly found in many programming languages. Loops are used for the purpose of providing a simple and easy way to repeat an action a counted number of times. In programming terminology, program code that repeats (executes again and again) is said to *iterate*.

## Using a Loop

To create a loop in a program, the Loop tile is dragged into the editor. When the Loop tile is dragged into the editor, a popup menu offers a choice for the **count** (the number of times the loop will execute), as shown in Figure 7-1-4. In this example, 8 was entered as the count.

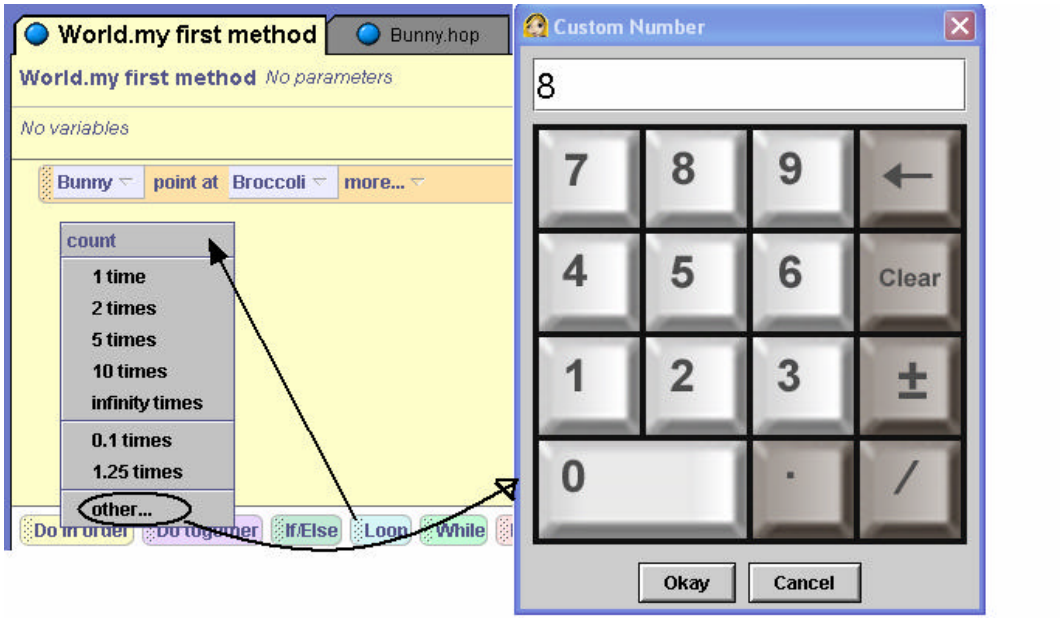



**Figure 7-1-4. Selecting a count**

Then, the *Bunny.hop* instruction is placed inside the Loop block, as shown in Figure 7-1-4. When the program is run, the Bunny will point at the Broccoli and then hop eight times. That is, the **loop will iterate eight times**. The benefit of using a Loop is immediately obvious – the Loop is quick and easy to write. And, the Loop is easy to understand!

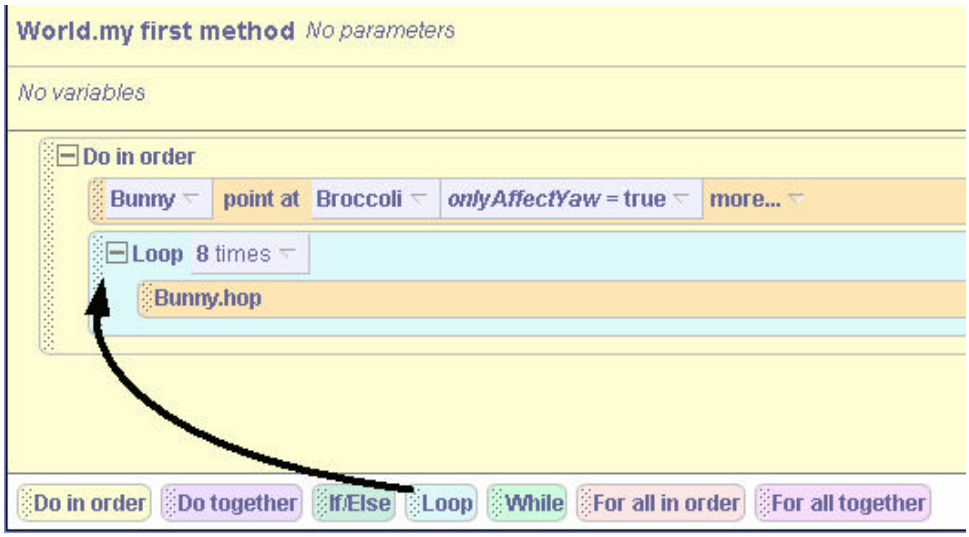



**Figure 7-1-4. Using the loop instruction**

## Count

In this example, the loop count tells Alice to run the *Bunny.hop* method 8 times. The count must be a positive number value. One of the options in the popup menu for selecting the count is *infinity*. (See Figure 7-1-3.) If infinity is selected, the loop will continue on and on until the program is stopped. In this example, if infinity were selected the bunny would never stop hopping until the user stops the program execution!

## Loops and Do in order or Do together

We placed a single instruction (Bunny.hop) in the Loop block for this example. But, several instructions could have been dragged into the block and all the instructions would be repeated. A *Do in order* or *Do together* block can be nested inside the loop to control how the instructions are repeated. If we do not use a *Do in order* or *Do together* block inside the Loop, Alice will assume that the instructions are to be done in sequential order. The following example will uses *Do together* blocks of instructions nested inside Loops.

## Nesting of loops

Consider the following situation. Suppose an initial scene with a two-wheeled Ferris wheel, as in Figure 7-1-5. An animation is to be written that simulates the running of this Ferris wheel. In this ride, to make it as jarring as possible, the double wheel will turn clockwise (rolling right in Alice terminology) while each of the individual wheels within the Ferris wheel turns counterclockwise.

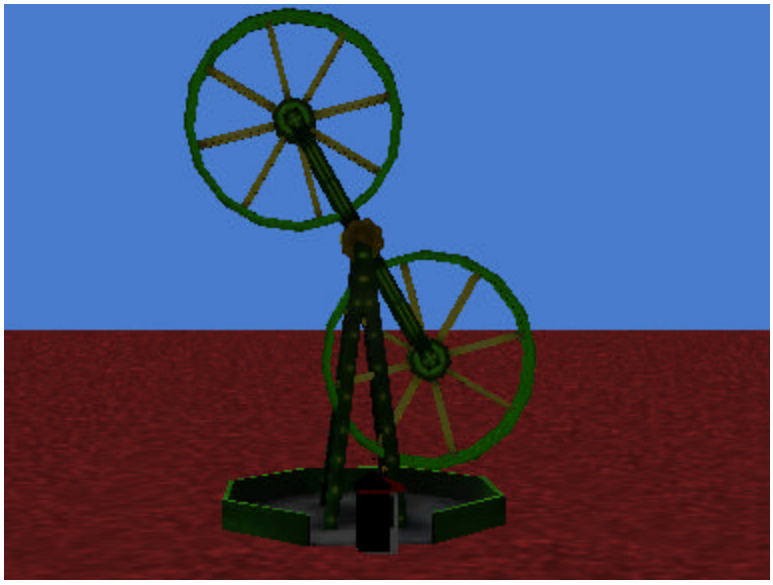


**Figure 7-1-5. A Ferris wheel**

Figure 7-1-6 is code for a Loop to roll the Ferris wheel ten times. The "*style=abruptly*" parameter has been used to smooth out the rotating of the entire double wheel over each iteration. (It is worthwhile to experiment with the different *style=* options whenever an animation happens to run choppy, as this one will run if the *style=abruptly* option is not used. The reason we chose the abrupt style is the default style, *gently*, slows down the animation instruction as it begins and ends. Because we are repeating the instruction within a loop, we do not wish to have a slowdown in the action.)

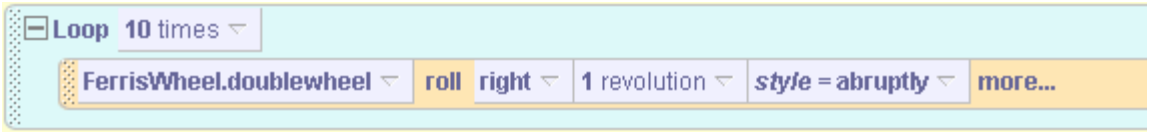



**Figure 7-1-6. Rotating the double wheel clockwise 10 times**

With the motion of the entire Ferris wheel accomplished, instructions can be written to rotate each of the inner wheels counterclockwise (rolling left) while the outer double wheel is rotating clockwise. The code to rotate the inner wheels is presented in Figure 7-1-7.

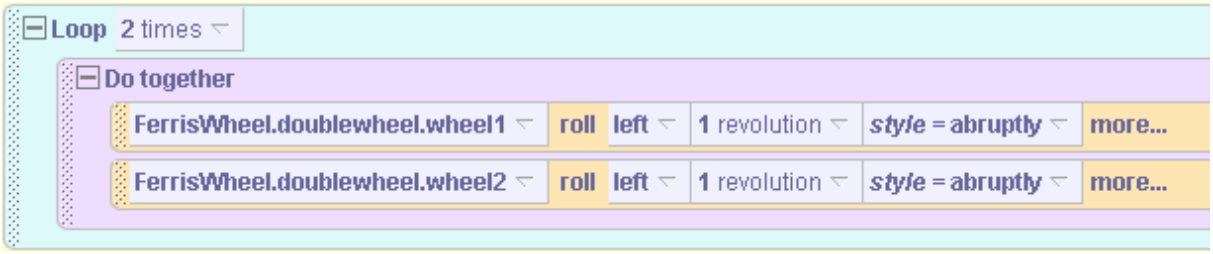



**Figure 7-1-7. Rotating the inner wheels clockwise 2 times**

Now, the rotations can be combined. Since each of the inner wheels has a smaller diameter than the outer double wheel, they should rotate more frequently, perhaps twice as often as the outer wheel. So, each time the double wheel rotates one revolution clockwise, the inner wheels should

at the same time rotate twice counterclockwise. Figure 7-1-8 shows the modified code. Note that, since the inner wheels need to rotate twice, the duration of the double wheel is made to be twice the duration of rotation of the inner wheels.

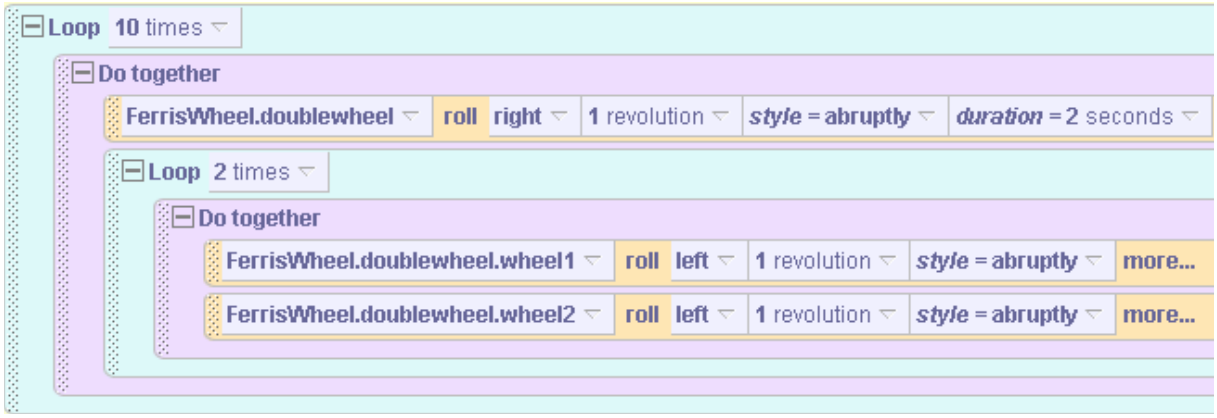



**Figure 7-1-8. The complete code for the Ferris wheel**

It is interesting to note the operation of the nested (inner loop). Each time the outer loop is run once, the inner loop runs twice. Notice that double wheel takes two seconds to complete its rotation, and the inner wheels will require 1 second (the default duration) to complete their rotations, but will rotate twice. In all, the inner wheels will rotate 20 times counterclockwise as the double wheel rotates 10 times clockwise.

### Technical note on looping

In the examples presented here, the number of times a block of code was to loop was a specific number, 2, 8, or 10. But the count can also be a question that returns a number. For example, if the boy is 6 meters from the girl, and we were to write the following loop:


The boy would turn a cartwheel 6 times.

**7-1 Exercises**

**1. Caught in the act**

This exercise is to complete the Bunny in the garden example in section 7-1. You will recall that in the storyline the baby bunny has snuck into his neighbor's garden and is hopping over to take a bite out of the tempting broccoli shoots. Code was presented to make the Bunny hop eight times (in a loop) over to the broccoli. But, we did not complete the example. Just as the bunny reaches the broccoli papa rabbit appears at the gateway to the garden and taps his foot in dismay. The bunny lowers his head sadly and hops a quick retreat out of the garden. Write a program to implement the Bunny in the garden animation. Your code should not only use a loop to make the bunny hop over to the broccoli (see the initial scene in Figure 7-1-1) but also to hop out of the garden when papa rabbit catches him in the garden.

**2. SquareBunnyHop**

This exercise is to explore the use of nested loops. Papa rabbit has been teaching the Bunny some basic geometry. The lesson this morning is on the square shape. To demonstrate that the Bunny understands the idea of a square shape, the bunny is to hop in a square. Create a world with the Bunny and the hop method, as described in section 7-1. Use a loop to make the Bunny hop three times. When the loop ends, have the Bunny turn left ¼ revolution. Then add another loop to repeat the above actions as shown in the storyboard below.

```
loop 4 times
    loop 3 times
        Bunny.hop
    turn left ¼ revolution
```

**3. BlimpD**

Create a scene as shown below with a blimp and a dragon. The dragon was on his way to the castle when he noticed a blimp. Dragons are rather curious creatures and this dragon can't resist going over to check it out. The dragon decides fly over and take a look around the blimp. Write a

program to have the dragon move to the blimp and then fly (length-wise) around the blimp three times. Your program must include methods for **MoveToBlimp** and **FlyAroundBlimp** and a loop to repeat the FlyAroundBlimp method three times.

**4. SnowManToStool**

This exercise is to practice using a number question as the count for a loop. Create a world with a snowman and a stool, as seen below. Use a loop to make the snowman move to the stool, stopping just short of bumping into the stool. Use a *distance to* question to determine the count (the number of times the loop repeats).

**5. SaloonSign**

This old saloon is being converted into a tourist attraction. Use 3D text (see Tips & Techniques 1 for details on using 3D text) to create a neon sign to hang on the front of the balcony. Then use a loop to make the sign blink 10 times.

## 7 Summary

The counted loop was introduced in this chapter as a simple construct for repeating an instruction or block of instructions. The counted loop is the second kind of control statement we have studied. We wrote our first program using a Loop instruction to make the Bunny hop eight times without having to drag the Bunny.hop instruction into the editor eight times. The advantage of this construct became immediately obvious. It was fast and easy to write and also easy to understand. Of course, it is possible to write Loop instructions that are much more complicated. But, overall the counted loop is an impressive programming tool.

### Important concepts in this chapter

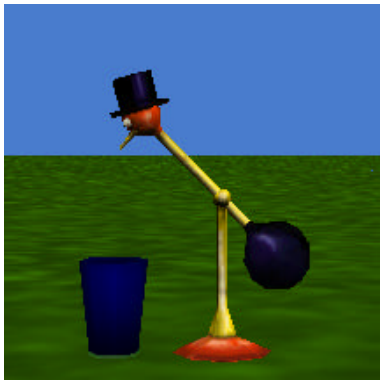
- The *Loop* instruction can be used to repeat an instruction, or a block of several instructions.
- A key component in a Loop is the *count* – the number of times the instructions within the Loop will be repeated.
- The count must be a positive number or infinity.
- If the count is infinity, the loop will repeat until the program shuts down.
- A Do in order or Do together can be nested inside a loop. If neither a Do in order or a Do together is placed within the loop, Alice assumes that the instructions are to be executed in order.
- Loops can be nested within loops.
- When a loop is nested within a loop, the inner loop will execute the inner-loop-count for each execution of the outer loop. For example, if the outer loop count is 5 and the inner loop count is 10 then the inner loop executes 10 times for each of the 5 executions of the outer loop. In other words, the outer loop would execute 5 times and the inner loop will execute 50 times.

## 7 Projects

### 1. DrinkingParrot

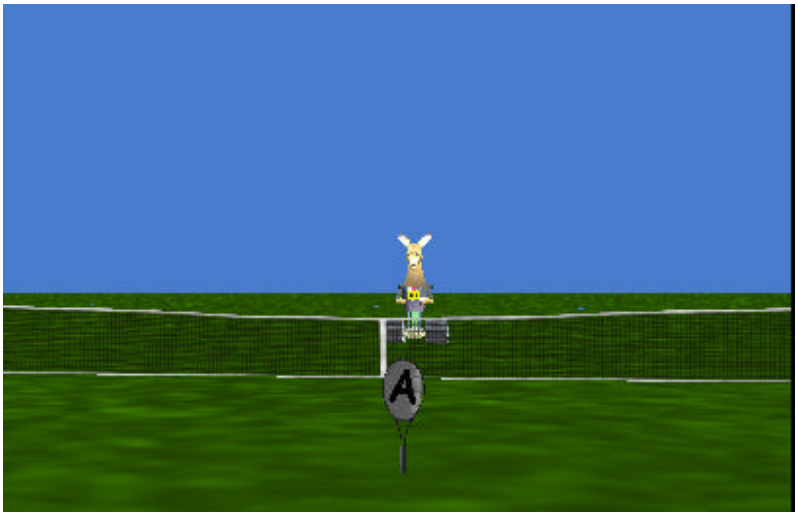
A small toy popular with children is a drinking parrot. The parrot is positioned in front of a container of water and the body of the parrot given a push. Because of the counterbalance of weights on either end of its body, the parrot repeatedly lowers its head into the water. Create a simulation of the drinking parrot (Objects folder in the gallery). Use an infinite loop to make the parrot drink.

*Hint:* In the world shown below, we used the blender object (Objects folder), pushed the base into the ground, and changed the color of the blender to blue to simulate a bucket of water.



### 2. TennisShot

Create a tennis shot game with a kangaroo_robot (SciFi folder on web gallery), a tennis ball, tennis net, and tennis racket (Sports folder on web gallery). Position the tennis ball immediately in front of the robot on the other side of the net (from the camera point of view) and the tennis racket on this side of the net (once again, from the camera point of view). The initial scene should appear as shown below.



Set up an event in the events handler to *let the mouse move* the tennis racket. (See Tips & Techniques 5) In this game, the kangaroo_robot and the tennis ball move together left or right a

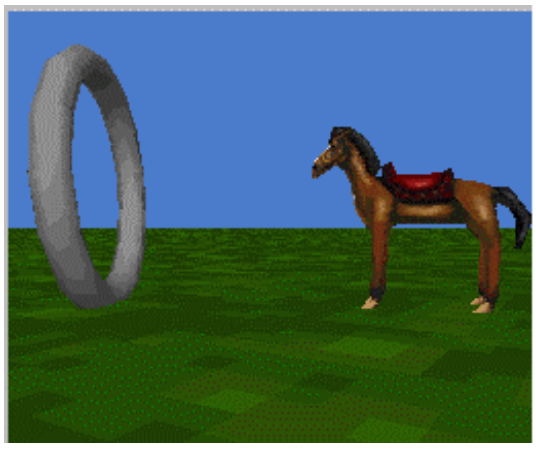
random distance (use the World random number question) between -1 and 1 meter. Then the robot "throws" the ball across the net -- the tennis ball moves up a random height and forward a given distance over a given duration. You will need to use trial and error to figure out an appropriate distance for the ball to move forward and the amount of time to allow for the move. When the ball has moved far enough towards the camera, it should be out of sight.

The player (user) will move the tennis racket to try to "hit" the tennis ball. A "hit" occurs when the tennis racket gets within 0.1 meters of the tennis ball. Actually, the tennis ball is virtual in this simulation and will go right through the racket even if the player manages to "hit" it. However, we will know if the player manages to "hit" the tennis ball, because the kangaroo_robot will wiggle his ears.

The real challenge in writing this program is to figure out whether the player is successful in moving the tennis racket to get it close enough to the tennis ball to hit it before it goes out of sight. To make this work, use a loop where each execution of the loop moves the tennis ball up and forward only a very short distance (something like 0.1 meters). Each time through the loop, check to see if the tennis racket has gotten close enough to the tennis ball to score a hit. As mentioned above, you will need to experiment with your world to figure out the appropriate count for the loop so as to eventually move the ball forward out of sight of the camera. When the loop ends, have the kangaroo_robot turn left ¼ revolution and move off the tennis court signaling that the game is over.

**3. TorusAndHorseTrot**

The horse is in training for a circus act. As part of the act, the horse is required to jump through a large hoop (a torus). Create a world, as shown below, with a horse facing the torus. Write a program to use a loop to have the horse **trot** forward several times so as to move through the torus. Use trial and error to determine the count for the loop.



Your project must include a **trot** method that makes the horse perform a trot motion forward. A horse trots by moving the right front leg forward at the same time the left leg moves forward and then moves the left front leg forward at the same time the right leg moves forward. The leg motions should bend at knee for a more realistic simulation. Of course, the leg motions should happen simultaneously with the entire horse moving forward. The loop should call the **trot** method a given number of times.

**4. Juggling**

Create a world with a character (you choose the character) and three juggling balls. Write a method to animate the character juggle the balls in the air. Use a loop to make the juggling act repeat five times and then the juggling balls should fall to the ground. Of course, the character must have at least two arms and be able to move them in some way that resembles a tossing motion.

*Hint:* A juggling ball is easily created by adding a sphere or a tennis ball to the world, resizing it, and giving it a bright color.