1

Conditional Loops

Conditions are a powerful tools for expressing complex programs. However, conditions are not enough. Some programs need to combine loops and conditions. Loops and conditions are combined in conditional loops, that is loops that execute a block of expressions while a certain condition holds. This chapter presents the conditional loops offered by Smalltalk and go over simple examples. We will use a lot conditional loops to simulate animal behavior in Chapter **??**.

1 Conditional Loops

The idea behind conditional loops is that a block (a sequence of expressions) is repeated as long as a certain condition holds. Smalltalk defines two messages, whileTrue: and whileFalse:, that allow you to define conditional loops as shown below.

An Example. Let's take a simple example to illustrate their use. Imagine we want a robot to move north until its y coordinate is smaller than 100 pixels as shown in Figure 1.1. A solution using a conditional loop is shown by the method 1.1; it can be invoked as shown in script 1.1.

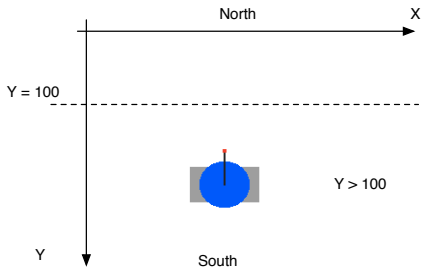

Figure 1.1: The situation for the execution of the method upTo100.

Script 1.1 (*Invoking upTo100*)

```
| pica |
pica := Bot new.
pica upTo100
```

Method 1.1

```
upTo100
  "Make forward the receiver until its ordinates is smaller than 100"

  self north.
  [ self center y > 100 ]
    whileTrue: [ self go: 10 ].
  self color: Color green.
```

Let's look carefully at what's happen when this method is executed.

1. The expression self north is not part of the conditional loop, so it is executed once.

2. The conditional loop is composed of a *condition* and a *conditional block* as shown in Figure 1.2. The condition which is expressed as a block [self center y > 100] is executed.

3. whileTrue:, the method loop name, defines the meaning of the loop: when the result of the condition is true and only then the conditional block, [self go: 10], is executed. Once the execution of the conditional block terminates the process restarts at step 2.

4. When the result of the condition [self center y > 100] in step 2 is false, the conditional block is *not* executed and the loop stops. The messages following the conditional expression are executed: here the expression self color: Color green gets executed and the method terminates. The program goes on to step 5.

5. The expressions following the conditional expression are executed. In the example, the expression self color: Color green gets executed and the method terminates.

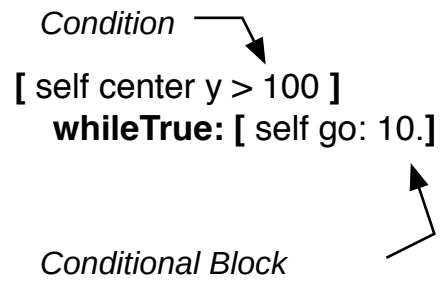

Figure 1.2: A whileTrue: conditional loop is composed of a condition and a conditional block. The conditional block is only executed when the condition is true.

A conditional loop is composed of a condition and a conditional block (that is a sequence of expressions).

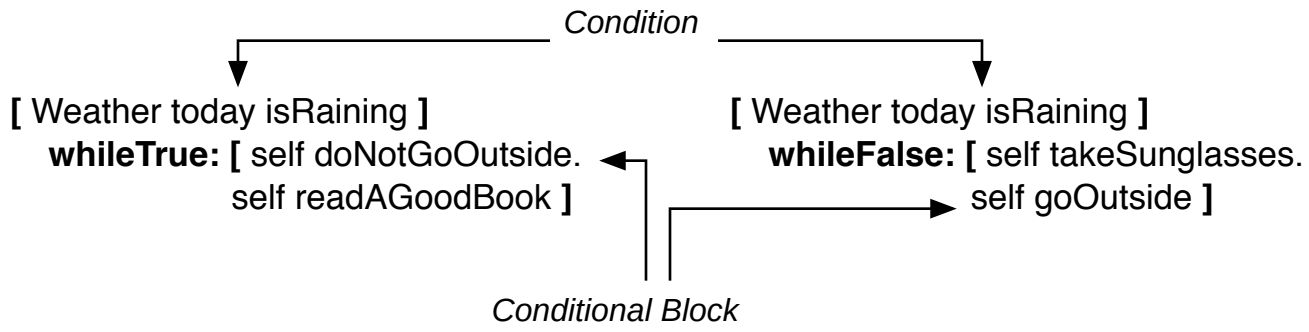

Figure 1.3: The whileTrue: and whileFalse: conditional loops are composed of a condition and a conditional block. whileFalse's conditional block is executed when whileFalse's condition is false. WhileTrue's conditional block is executed when whileTrue's condition is true.

The meaning of the loop is defined by its method name: whileTrue: executes its condition and executes the conditional block when the condition is true. whileFalse: does the same but executes the conditional block only when the condition is false.

@@— the following should be in a frame—@@ whileTrue: and whileFalse: allow you to define conditional loops that is the conditional block is repeated while the condition holds.

```
[ condition ] whileFalse:
  [ conditional messages ]

[ condition ] whileTrue:
  [ conditional messages ]
```

@@— until here—@@

2 Experiences with Traces

I suggest you add traces in the method you just wrote and analyze the resulting trace. Use also the debugger. Do not hesitate to modify its location in the method. The place where you insert the trace generation has also an impact on the resulting trace.

For example in method 1.2 I introduced a trace inside the conditional message before they get executed. I obtained the trace shown in Figure 1.4.

Method 1.2

```
upTo100
  "Make forward the receiver until its ordinates is smaller than 100"

  self north.
  [ self center y > 100 ]
    whileTrue:
      [Transcript show: '* ' , self center y asString.
      self go: 10 ].
  self color: Color green
```

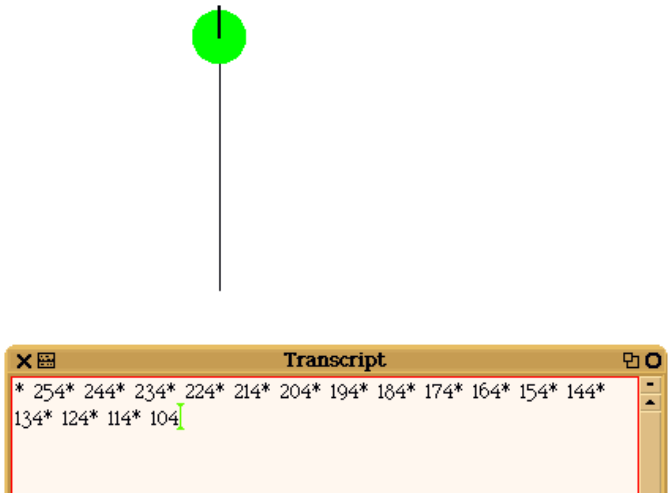

Figure 1.4: The trace of the execution of the method upTo100.

As an alternative experience, introduce the line Transcript show: '* ' , self center y asString ; cr. after the expression self go: 10 as shown by method 1.4 or even in the condition before the first line as shown by method 1.4.

Method 1.3

```
upTo100
  "Make forward the receiver until its ordinates is smaller than 100"

  self north.
  [Transcript show: 'c ' , self center y asString; cr.
  self center y > 100 ]
    whileTrue:
      [Transcript show: '* ' , self center y asString; cr.
      self go: 10 ].
  self color: Color green
```

Method 1.4

```
upTo100
  "Make forward the receiver until its ordinates is smaller than 100"

  self north.
  [ self center y > 100 ]
    whileTrue:
      [ self go: 10
      Transcript show: '# ' , self center y asString; cr ]
```

Compare the traces produced by the different methods. Look in particular at the final values.

whileTrue: can be converted to whileFalse: by negating (logically reverting) the condition. Use whichever method helps you understand your program better. Try to define the method upTo100 that uses whileFalse: instead of whileTrue:. Then define a method that makes move one pixel at a time. Compare the exact position where it stops.

Sometimes defining correct conditional loops is difficult. It is easy to forget to check the condition carefully, and the loop can repeat endlessly. While writing a conditional loop you should always keep in mind that the loop should somehow tend toward the end of the loop that is tend toward a situation that makes the condition not hold anymore.

Try this experiment: move the robot by using the black halo close to the top edge of the window, that is be sure its y coordinate is smaller than 100. Then invoke the method upTo100. As you see, nothing happens. This as expected: the method is invoked and the expression self center y > 100 is false, because the position of the robot is smaller than 100. Therefore the conditional block is not executed.

3 Stopping an Infinite Loop

It is not exceptional to write an endless loop. Practically, this happens because the condition expression never returns true for whileFalse:; and for whileTrue: it is because the condition never returns false.

If you do write an endless loop, you can stop it by pressing Apple-. on Mac or Alt . or Control-C on other platforms. Then to understand why the loop did not terminate you can use the debugger that pops up by clicking its Debug button. You can also print some information on the transcript and analyze it.

A conditional loop can endlessly loops when the conditional block does not perform an action that will eventually breaks the condition that is stop the condition from holding anymore.

Let's consider this difficult point in the case of our example. The distance between the robot and the vertical line having 100 as y value gets smaller and smaller. In the example, to be sure that the loop gets a chance to terminate, the conditional messages should somehow reduce this distance. As long as the robot center's y coordinate is too big (>= 100), the loop will continue. So to be sure that the loop will terminate, the conditional block must somehow reduce the robot's y coordinate. The expression self go: 10 in the conditional block does this, because the y coordinate gets smaller as the robot moves north.

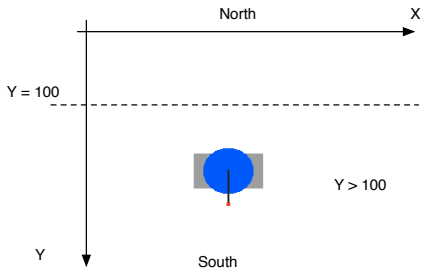

Figure 1.5: The situation for the execution of the method upTo100Infinite.

Let's look at the case shown in method upTo100Infinite (1.5). Here the condition is to stop the loop when the y position of the robot is smaller than 100, that is if the robot arrives above the horizontal line at 100 pixels from the top of the screen. However as the robot is going to the south as shown in Figure 1.5, there is no way that its y coordinate may go below 100. The method 1.5 cannot terminate because the conditional block cannot change the condition to be false. Here, the conditional block increases the value of the y coordinate, therefore the chance that the condition may evaluate to false reduces each repetition. This example is exaggerated but it illustrates clearly the problem of specifying loops that terminate.

Method 1.5

```
upTo100Infinite
  "Make forward the receiver until its ordinates is smaller than 100"

  self south.
  [ self center y > 100 ]
    whileTrue: [ self go: 10 ].
  self color: Color green
```

> When you are defining a loop always ask yourself if there is a possibility that the condition might never be met. This seems obvious but if the condition does not have a chance to fail the loop will never finish.

4 Deeper into whileTrue: and whileFalse:

The condition of a conditional loop does not have to contain just a single expression. It can contain a sequence of expressions as long as the last message of the condition returns either true or false. This allows one to express more complicated conditional loops.

Important Messages 1.1

```
[ self doThis.
anObject doThat.
self isStillWorking ] whileTrue:
  [self grumbleAndKeepOnWorking ]
```

Therefore you could change the method upTo100 to be as method 1.6. While this method looks nearly the same as upTo100, it has sometimes a different effect. Try to understand what the difference is. For example, add a trace or invoke the debugger in the method 1.6 and analyze it.

Method 1.6

```
notTheSameUpTo100
  "Move the receiver north until its y coordinate is smaller than 100"

  self north.
  [ self go: 10.
  self center y > 100 ]
    whileTrue: [ ].
  self color: Color green
```

As the condition is always executed at least once, the main difference is that the robot will move even if it already has a y coordinate smaller than 100.

5 A Simple Application

Conditional loops can be used to ask some values to the user until he gives a correct value. Imagine that you want to ask a user for the number of steps that you should draw. A solution without using a conditional loop is shown in method 1.2. First you prompt the user to get a string that represents the number of steps. Then you check whether the data that the user entered represents a number with the isAllDigits. When this is the case with convert the string into a number using the method asNumber and you perform the loop.

Script 1.2 (*Interactive stair)*

```
| pica  |
answer := (FillInTheBlank
                    request: 'Number of steps'
                    initialAnswer: '10').
answer isAllDigits
  ifTrue: [ pica := Bot new.
        answer asNumber timesRepeat:
             [ pica
               go: 10 ;
               north ;
               go: 10 ;
               east ]
```

Now you can force the user to enter a value that is a number by using a conditional loop that is repeatedly asking for a value if the previous value was not a number. Script 1.3 shows such a version.

Script 1.3 (*Interactive stair)*

```
| pica answer |
[ answer := (FillInTheBlank
                request: 'Number of steps'
                initialAnswer: '10').
answer isAllDigits] whileFalse: [ ].
pica := Bot new.
answer asNumber timesRepeat:
  [ pica
    go: 10 ;
    north ;
    go: 10 ;
    east ]
```

6 When to Use [ ]

Now that I presented conditions which require the use of square brackets [ ] at different places, you may have difficulties remembering when to put squared brackets [ ]. There are basically two rules in Smalltalk. You surround an expression or a sequence of expressions with [and] when:

- You need to execute several times the same expression. For example,
  - 4 timesRepeat: [pica go: 10; turnLeft:90] repeats 4 times the messages pica go: 10; turnLeft:90,
- The expression is executed a variable number of times. For example,
  - distance < 200 ifTrue: [self color: Color red] only executes self color: Color red under certain circumstances,
  - [self center y > 100] whileTrue: [self go: 10] repeats multiple times conditionally both self center y > 100 and self go: 10, therefore the receiver and the argument are blocks.

Note that this is an approximation since if you write 1 timesRepeat: [self go: 120] you still need to put bracket around the expression self go: 120. But this is a nice trick to remember. The real definition is: the argument of a conditional message (ifTrue:, ifFalse:, ifTrue:ifFalse:) or conditional loop message (whileTrue:, whileFalse:) is enclosed in square brackets. The receiver of a conditional loop message (whileTrue:, whileFalse:) is enclosed in square brackets.

7 Summary

- A conditional loop is composed of a condition and a conditional block that is a sequence of expressions.
- whileTrue: and whileFalse: allow you to define conditional loops that is the conditional block is repeated while the condition holds.
- When you are defining a loop always ask yourself if there is a possibility that the condition might never be met. This seems obvious but if the condition does not have a chance to fail the loop will never finish.

- Surround an expression with [ ] when (1) you need to execute several times the same expression (4 timesRepeat: [self go: 10]) or (2) the expression is not always executed (dist < 200 ifTrue: [self color: Color blue]).

Method	Description
[aCondition **] whileFalse:** **[** SequenceOfMessages **]**	Execute aCondition; if it is false execute SequenceOfMessages and repeat this step. If aCondition is true, pass to the next expression without executing SequenceOfMessages.
[aCondition **] whileTrue:** **[** SequenceOfMessages **]**	Execute aCondition; if it is true execute SequenceOfMessages and repeat this step. If aCondition is false, pass to the next expression without executing SequenceOfMessages.