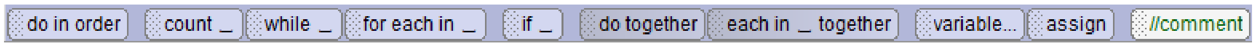
Control Structures

The goal of this lesson is to learn to use control structures such as loops to create 3D animation.

If you look at the tiles that appear at the bottom of the code editor, you will see the following tiles: do in order, count, while, for each in, if, do together, each in together, variable, assign, and comment.



You already know how to:

- to add comments using the comment tile;
- declare variables using the variable tile;
- give a variable a new value with the assign tile.

The rest are control structures. They allow you to control the order that statements are executed. Let's talk about each of these.

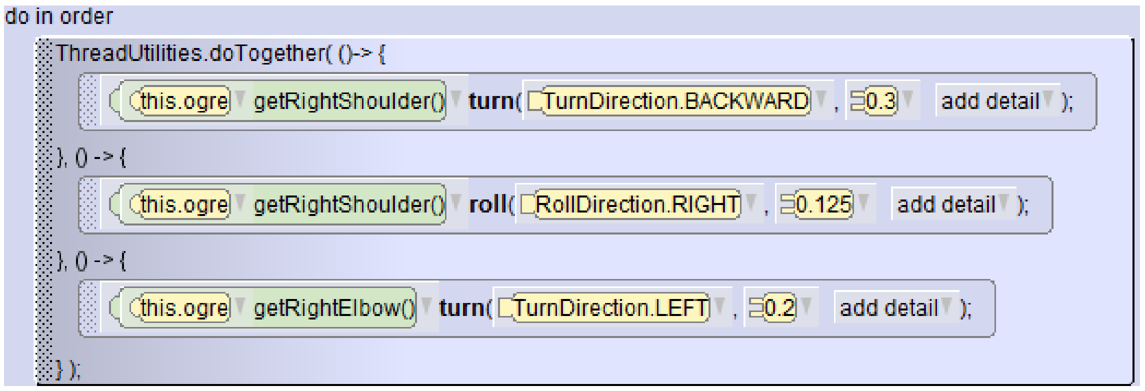
do in order

When we go to our code editor, we can see that the statements in myFirstMethod say "do in order." That means that the statements will be executed one after the other in sequence. "Do in order" is the default control structure that is used.

do together

If we have a character first turn his shoulder, then his elbow, to raise his arm, the movement looks a bit robotic. If you raise your arm you turn your shoulder and elbow at the same time.

If you drag the do together tile into the code, you can drag the statements for the shoulder and the elbow into the control structure and the movement will look much more natural. Here Thor's arm is positioned up ready to wave.

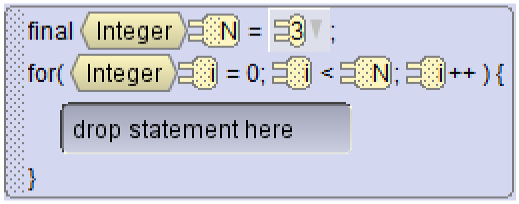


```
do in order
  ThreadUtilities.doTogether( ()-> {
      this.ogre getRightShoulder() turn( TurnDirection.BACKWARD , 0.3  add detail );
  }, () -> {
      this.ogre getRightShoulder() roll( RollDirection.RIGHT , 0.125  add detail );
  }, () -> {
      this.ogre getRightElbow() turn( TurnDirection.LEFT , 0.2  add detail );
  } );
```

count

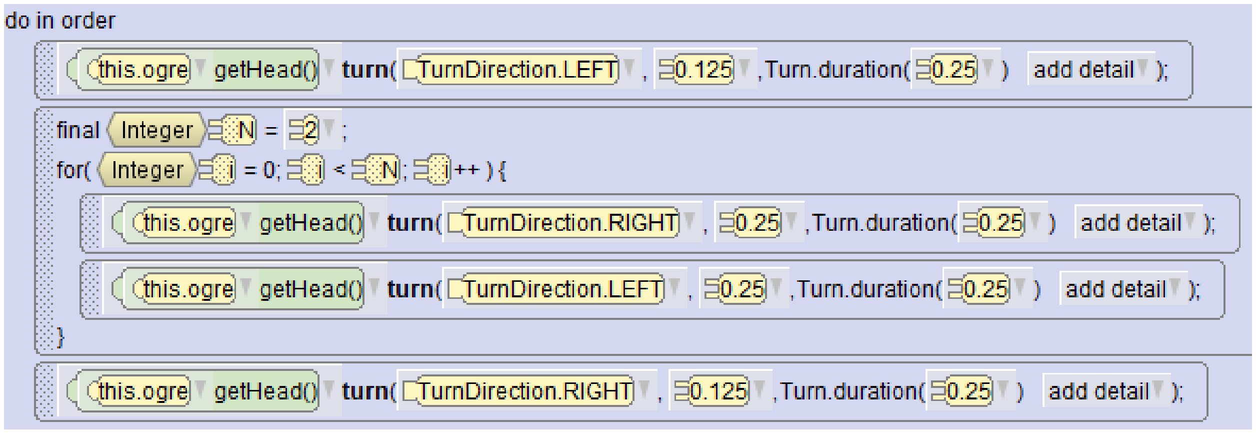
The count control structure lets us specify that we want to do something a fixed number of times. For instance, we might want to have a character shake their head no 2 times.

When we drag the count tile into the code, you will be asked to select an integer for the number of times we want the statements inside the control structure to execute. The code now looks as shown below. Notice that a constant N has been given a value of 3. The for loop uses a varialbe i to count. The **for** statement has 3 parts: the initial value of 0; a test for when the loop will keep going (as long as i is less than N) and an increment, i++ that will add 1 to i so that the value of i will eventually be 3 and the loop ends. The values of i will be 0, 1, 2. When i reaches N the loop ends. Any statements we put inside the loop will be executed N times.



```
final Integer N = 3 ;
for( Integer i = 0; i < N; i++ ) {
    drop statement here
}
```

To make a character shake his head 3 times, we could drag the procedure to turn the head to the left and then turn the head to the right into the count loop.



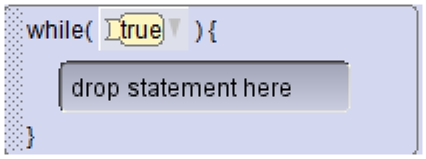
```
do in order
  (this.ogre getHead() turn( TurnDirection.LEFT , 0.125 ,Turn.duration( 0.25 ) add detail );
  final Integer N = 2 ;
  for( Integer i = 0; i < N; i++ ) {
    (this.ogre getHead() turn( TurnDirection.RIGHT , 0.25 ,Turn.duration( 0.25 ) add detail );
    (this.ogre getHead() turn( TurnDirection.LEFT , 0.25 ,Turn.duration( 0.25 ) add detail );
  }
  (this.ogre getHead() turn( TurnDirection.RIGHT , 0.125 ,Turn.duration( 0.25 ) add detail );
```

In this code, the ogre first turn his head to the left 0.125. Then he moves it to right then the left 2 times. When the loop ends, we move his head back to the right so his head is back where it started. You usually need statement before the loop to get in position, then statements after the loop ends to return to the starting position.

In Alice, the **for** loop always counts from 0 to some limit. You cannot change this to count down or count by twos or some other increment. To do that you will need to use a while loop.

while

The while control structure lets us specify that we want to do something as long as some condition is true. When you drag the while tile into your code you must select either true or false (Boolean values can only be true or false.) Usually you will select true.

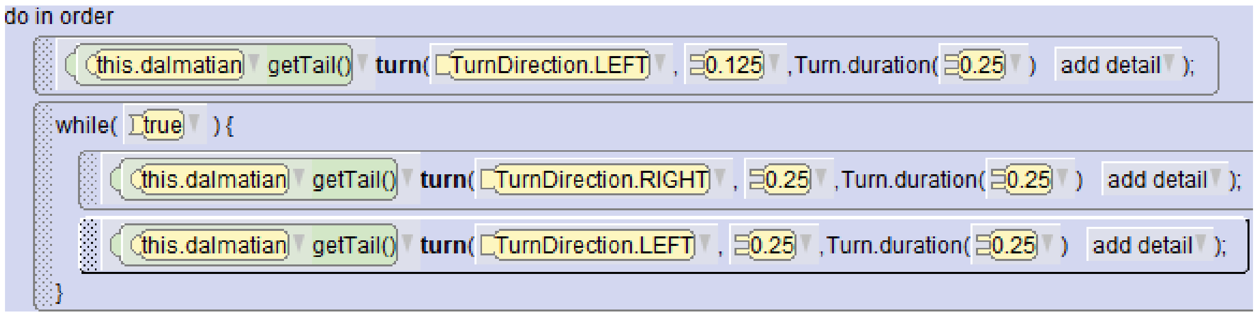


```
while( true ){
    drop statement here
}
```

You will learn more about Boolean expressions later. Then you might do something like have a shark move around while he hasn't caught a fish; or have the ogre walk while he isn't at the gate.

Any statements you drag into this control structure will execute over and over until the condition is false. If we use while(true) we will have an endless loop: it will never end.

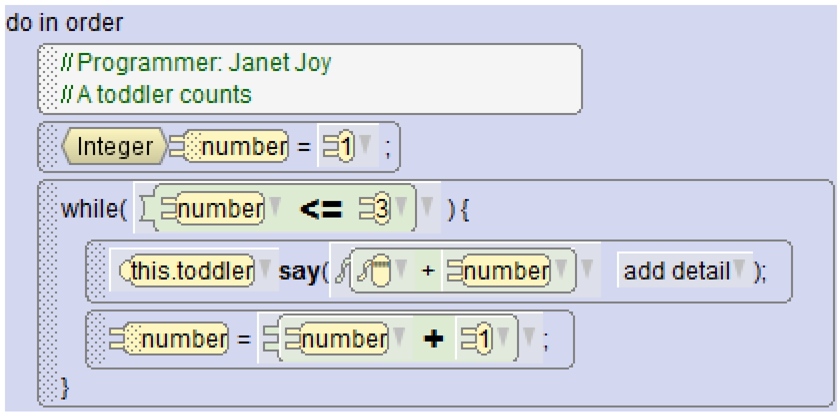
I have a scene with a Dalmatian and I want him to wag his tail continuously.



```
do in order
    (this.dalmatian getTail() turn( TurnDirection.LEFT , 0.125 ,Turn.duration( 0.25 ) add detail );
    while( true ){
        (this.dalmatian getTail() turn( TurnDirection.RIGHT , 0.25 ,Turn.duration( 0.25 ) add detail );
        (this.dalmatian getTail() turn( TurnDirection.LEFT , 0.25 ,Turn.duration( 0.25 ) add detail );
    }
```

The first statement turns the tail to the left to get it into position. There is no statement return the tail to its starting position when the loop ends because the loop will never end.

A while loop can be used if you want to count, but start with something other than 0. In this example the toddler will say 1, 2, 3.



```
do in order
    // Programmer: Janet Joy
    // A toddler counts
    Integer number = 1 ;
    while( number <= 3 ){
        this.toddler say( "" + number add detail );
        number = number + 1 ;
    }
```

if

The **if** control structure lets you execute one group of statements when a Boolean expression is true and another group of statements when the expression is false. We will discuss the if control structure in the lesson on Boolean expressions.

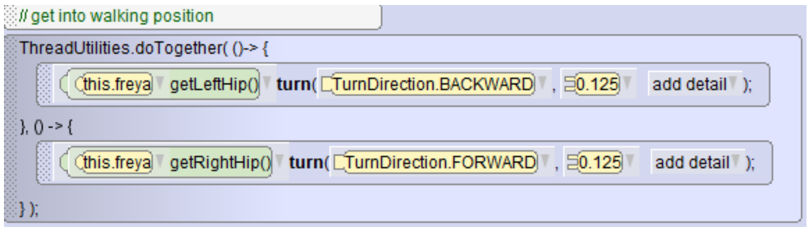
for each in and each in together

These two control structure let you work with arrays (lists). For instance you might want to create an array (list) of characters and have them each say hello one at a time using **for each in**, or have them all say hello together using **each in together**. We will discuss these two control structures in the lesson on Arrays.

Nesting Control Structures

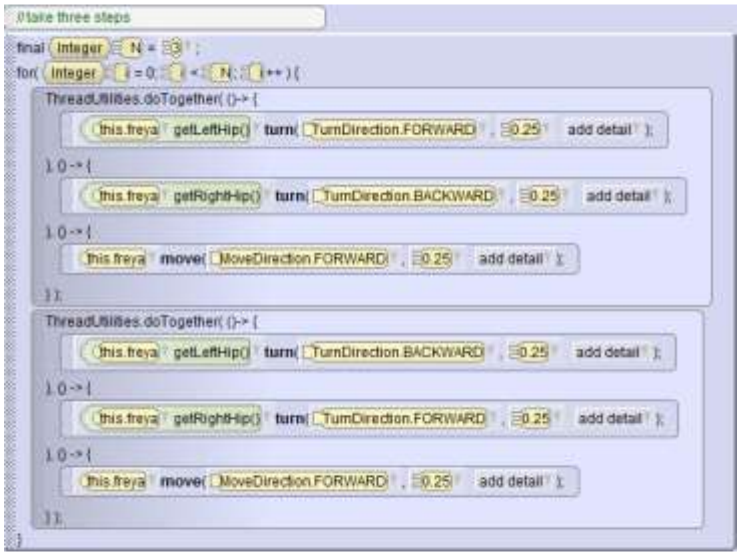
Some of the most interesting animation have multiple control structures, some nested inside each other. Walking is an example. The code here shows only the hip movement. When we add the movement of the shoulders, knees, ankles and elbows it becomes even more complex.

To get into position to start walking, we want the hips to move together. One leg goes back, the other goes forward. This block is repeated in reverse at the end to go back to the starting position.



```
// get into walking position
ThreadUtilities.doTogether( ()-> {
    this.freya getLeftHip() turn( TurnDirection.BACKWARD , 0.125 add detail );
}, () -> {
    this.freya getRightHip() turn( TurnDirection.FORWARD , 0.125 add detail );
} );
```

Taking 3 steps has a count loop with two doTogether control structures inside.



```
//take three steps
final Integer N = 3 ;
for( Integer i = 0; i < N; i++ ){
    ThreadUtilities.doTogether( ()-> {
        this.freya getLeftHip() turn( TurnDirection.FORWARD , 0.25 add detail );
    }, () -> {
        this.freya getRightHip() turn( TurnDirection.BACKWARD , 0.25 add detail );
    }, () -> {
        this.freya move( MoveDirection.FORWARD , 0.25 add detail );
    } );
    ThreadUtilities.doTogether( ()-> {
        this.freya getLeftHip() turn( TurnDirection.BACKWARD , 0.25 add detail );
    }, () -> {
        this.freya getRightHip() turn( TurnDirection.FORWARD , 0.25 add detail );
    }, () -> {
        this.freya move( MoveDirection.FORWARD , 0.25 add detail );
    } );
}
```