

Unit II: Creating Code

Chapter 13: Creating Animation in Alice3

Animation is created in Alice by moving objects a little bit at a time using several different control structures. In this chapter, you will learn how to use several important control structures in Alice to control the sequence and repetition of actions. You will use **doInOrder** to perform actions one after another and **doTogether** to perform actions at the same time. You will also work with **count** and **while loops** to repeat actions and learn how to combine these control structures to create more complex 3D animations. In addition, you will learn how to create and edit markers and use them in animation code.

Objectives

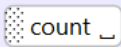
- Utilize control structures in your code.
- Determine which control structures to use for more natural movements.
- Implement loop control structures **count** and **while** to wave, clap, shake the head, and stomp the feet!
- Use markers to move an object to a set position.
- Edit object markers: add, delete, re-rename, and change the color;
- Create an animation that uses several subparts, such as walking.

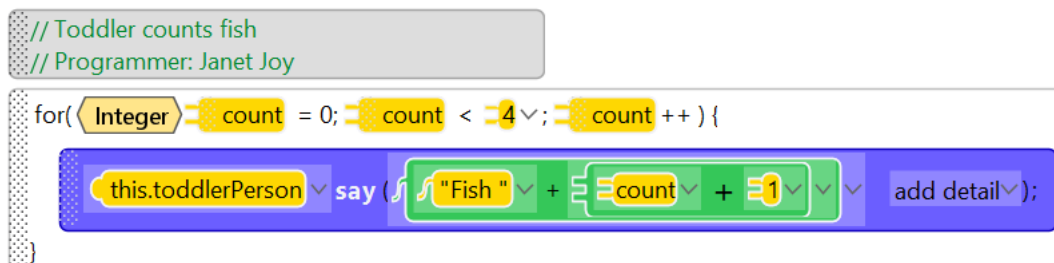
Control Structures

Control Structures are the blocks that control the order that statements are executed. It includes loops, do in order, do together, and if/else blocks.

Counting

In this scene there is a toddler in a boat and 4 fish in the water. The toddler will count them. We did this in the math lesson; there were only four fish, so it was simple enough to repeat the same code 4 times using the clipboard, but now there are 12. This time we will use a **count** loop.

When we drag the  count tile into our code, it creates a **for** loop:



The **for loop** uses an **integer** variable. The first part of the statement gives the variable an initial value: `count=0`; the second part is a **Boolean expression** telling when we

keep going: `count<4`; the third part tells what to add each time: `count+1`. This is called the **increment**.

Counting by 5

In the code below **indexA** (the default) is given a starting value of 0; the loop continues as long as **indexA <10**; **indexA++** means that 1 is added to the value of **indexA** for each iteration (repetition) of the loop.

```
// Loki counts by 5
// Programmer: Janet Joy

for( Integer indexA = 0; indexA < 5; indexA ++ ) {
    Integer num = indexA * 5;
    this.loki say ( "" + num add detail );
}
```

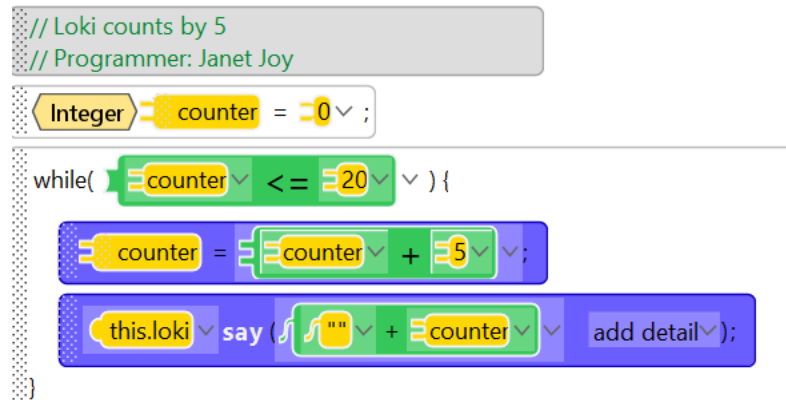
Let's trace through the code to see what happens:

indexA	num	Loki Says
0	0	0
1	5	5
2	10	10
3	15	15
4	20	20
5	indexA is not < 5, loop ends	

Basically, with the **for** loop, you tell it how many times to repeat. Because we start with a value of 0 and keep going as long as the value is less than the end point, the end point is the number of times the loop repeats.

Counting by 5

In the code below, we use a while loop. A while loop does not have a built-in variable, so we declare the integer counter before the loop. The while loop has one Boolean expression that tells when to keep going. Here, we say `while(counter<=20)` When a new iteration of the loop begins, it checks whether the Boolean expression is true or false. If it is true, the entire body of the loop executes, even if that Boolean expression becomes false inside the loop.



Let's trace through the code to see what happens:

Counter is initialized to 0

Test: $0 \leq 20$ **true:** execute the loop statements
 counter=count+5, counter is now 5

Loki says 5

Test: $5 \leq 20$ **true:** execute the loop statements
 counter=count+5, counter is now 10

Loki says 10

Test: $10 \leq 20$ **true:** execute the loop statements
 counter=count+5, counter is now 15

Loki says 15

Test: $15 \leq 20$ **true:** execute the loop statements
 counter=count+5, counter is now 20

Loki says 20

Test: $20 \leq 20$ **true:** execute the loop statements
 counter=count+5, counter is now 25

Loki says 25

Test: $25 \leq 20$ **false:** the loop ends.

(When the loop ends it continues with the statement after the loop.)

Endless Loops

A while loop can be endless if the **while condition** can never be **false**.

While(true) is endless. In the code below, the fish take turns counting but the loop never ends, so the statement where the clown fish says goodbye never executes.

```
// Fish count off
// Programmer: Janet Joy

Integer count = 0;

while( true ) {
    count = count + 1;
    this.clownFish say ( "" + count add detail);
    count = count + 1;
    this.pajamaFish say ( "" + count add detail);
    count = count + 1;
    this.carp say ( "" + count add detail);
    count = count + 1;
    this.blueTang say ( "" + count add detail);
}

// This statement will never execute, because the loop is endless!
this.clownFish say ( "Goodbye!" add detail);
```

We could make the loop end by changing **while(true)** to **while(count<3)**. Then the loop would execute just once, but the fish would say 1, 2, 3, 4, because once the loop started it would execute all the statements, even after count became 3.

Markers

In this scene there is an ice skater and a few objects that she wants to skate around. We can't move to an object using place, because that would put her pivot point at the objects pivot point, which would imbed her inside the object. Instead, we will create some markers for the position we want her to move to.



Object Markers

Select the skater, then click Add Object Marker. The starting position marker is named **startMarker**.

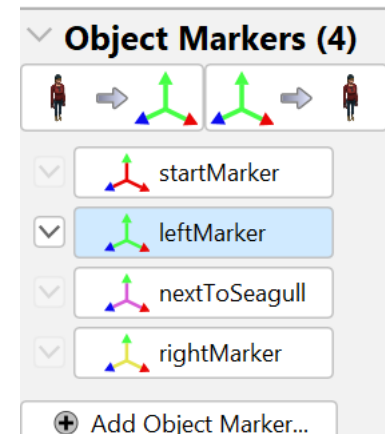
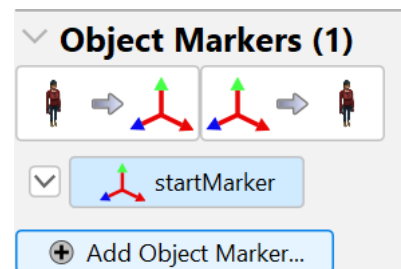
Now, rotate the skater to face the next position, then move her to that position and add another object marker, giving it a good name, such as **leftMarker**, **nextToSeagull**, etc. Continue adding markers until you have created an interesting sequence.

Edit Markers

Select one of the markers. There will be a drop-down arrow to its left. This will let you rename it, delete it or change its id color.

At the top of the list of markers, there are two images. The first image lets you move the skater to the selected marker. If you have rotated her position at each marker, you will see the animation of the skater turning as she moves to the new location. The second image lets you adjust the marker. Move the skater to make the adjustment, then click that second button to change it to the skater's current position.

You can select each of the markers to see what the skater will do as she moves to each marker.



Code Using Markers

```
// Ice Skater with Markers
// Programmer: Janet Joy

this.iceSkater turnToFace ( this.camera add detail );
this.iceSkater say ( "Welcome to My Skating Rink!" add detail );
this.iceSkater turn ( TurnDirection.LEFT, 2.0 add detail );
this.iceSkater moveAndOrientTo ( this.nextToSeagull add detail );
this.iceSkater turn ( TurnDirection.RIGHT, 2.0 add detail );
this.iceSkater moveAndOrientTo ( this.rightMarker add detail );
this.iceSkater moveAndOrientTo ( this.startMarker add detail );
this.iceSkater say ( "I hope you enjoyed the show!" add detail );
```

Add a Loop

We aren't counting anything, and she isn't saying the count, we just want her to repeat her routine a few times, so a **for loop** is perfect here.

```
void myFirstMethod ( )
do in order
  // Ice Skater with Markers
  // Programmer: Janet Joy
  this.iceSkater turnToFace ( this.camera add detail );
  this.iceSkater say ( "Welcome to My Skating Rink!" add detail );
  for( Integer indexA = 0; indexA < 3; indexA ++ ) {
    this.iceSkater turn ( TurnDirection.LEFT, 2.0 add detail );
    this.iceSkater moveAndOrientTo ( this.nextToSeagull add detail );
    this.iceSkater turn ( TurnDirection.RIGHT, 2.0 add detail );
    this.iceSkater moveAndOrientTo ( this.rightMarker add detail );
    this.iceSkater moveAndOrientTo ( this.startMarker add detail );
  }
  this.iceSkater say ( "I hope you enjoyed the show!" add detail );
```

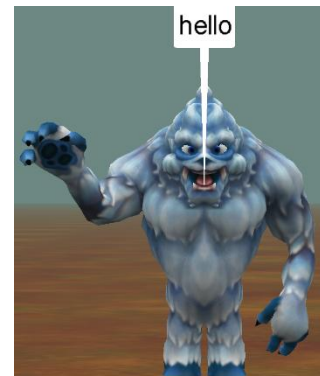
Experiment! Make your own skating routine. Make the bunny hop around, have a UFO fly from marker to marker, etc.

Waving

We have a simple scene with a Yeti facing forward, so that we can see the details. Then we make him wave.

There are three parts to the code:

1. Raise arm
2. Repeat wave motion 3 times
3. Lower arm back to starting position.



```

public void myFirstMethod() {
    // Yeti Waves
    // Programmer : Janet Joy
    Double turnBack = 0.2;
    // Get arm into position
    doTogether( () -> {
        this.yeti.say( "hello", Say.duration( 10.0 ) );
    }, () -> {
        this.yeti.getRightShoulder().turn( TurnDirection.LEFT, turnBack );
    }, () -> {
        this.yeti.getRightElbow().turn( TurnDirection.BACKWARD, turnBack );
    } );
    // Wave three times
    Double waving = 0.16;
    Double turnRight = 0.16;
    this.yeti.getRightElbow().turn( TurnDirection.LEFT, waving );
    for( Integer indexA = 0; indexA < 3; indexA++ ) {
        this.yeti.getRightElbow().turn( TurnDirection.RIGHT, turnRight );
        this.yeti.getRightElbow().turn( TurnDirection.LEFT, waving );
    }
    // Back to straight arm
    doTogether( () -> {
        this.yeti.say( "Bye" );
    }, () -> {
        this.yeti.straightenOutJoints();
    } );
}

```

Here we use `this.yeti.straightenOutJoints()`; this straightens all the joints. If he were sitting to start, we would need to put the arm back using the opposite of the code to get his arm in position:

```

this.yeti.getRightShoulder().turn( TurnDirection.RIGHT, turnBack );
}, () -> {
    this.yeti.getRightElbow().turn( TurnDirection.FORWARD, turnBack );
} );

```