ORACLE Academy

Using Control Statements and Functions

What Will I Learn?

Objectives

- Define multiple control statements to control animation timing
- Create an animation that uses a control statement to control animation timing
- Recognize programming constructs to invoke simultaneous movement
- Use functions to control movement based on a return value
- Create programming comments

Why Learn It?

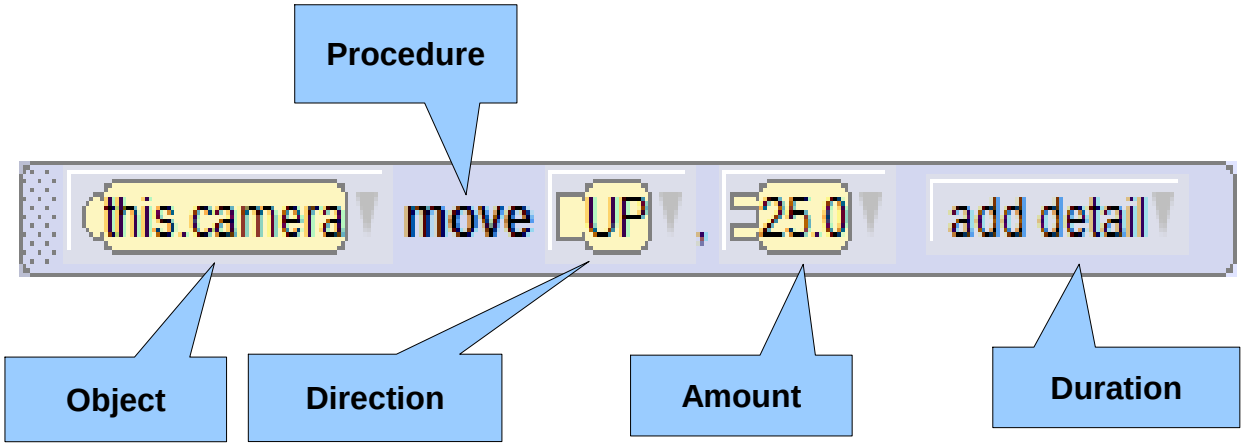
Purpose

Certain movements require coordination, such as walking, sitting, or riding in a vehicle. Coordination may be required for an object's body parts to move in sync, or for two objects to move relative to each other.

Control statements and functions will help you better coordinate the movements of objects, as well as make their movements more precise.

Editing Arguments

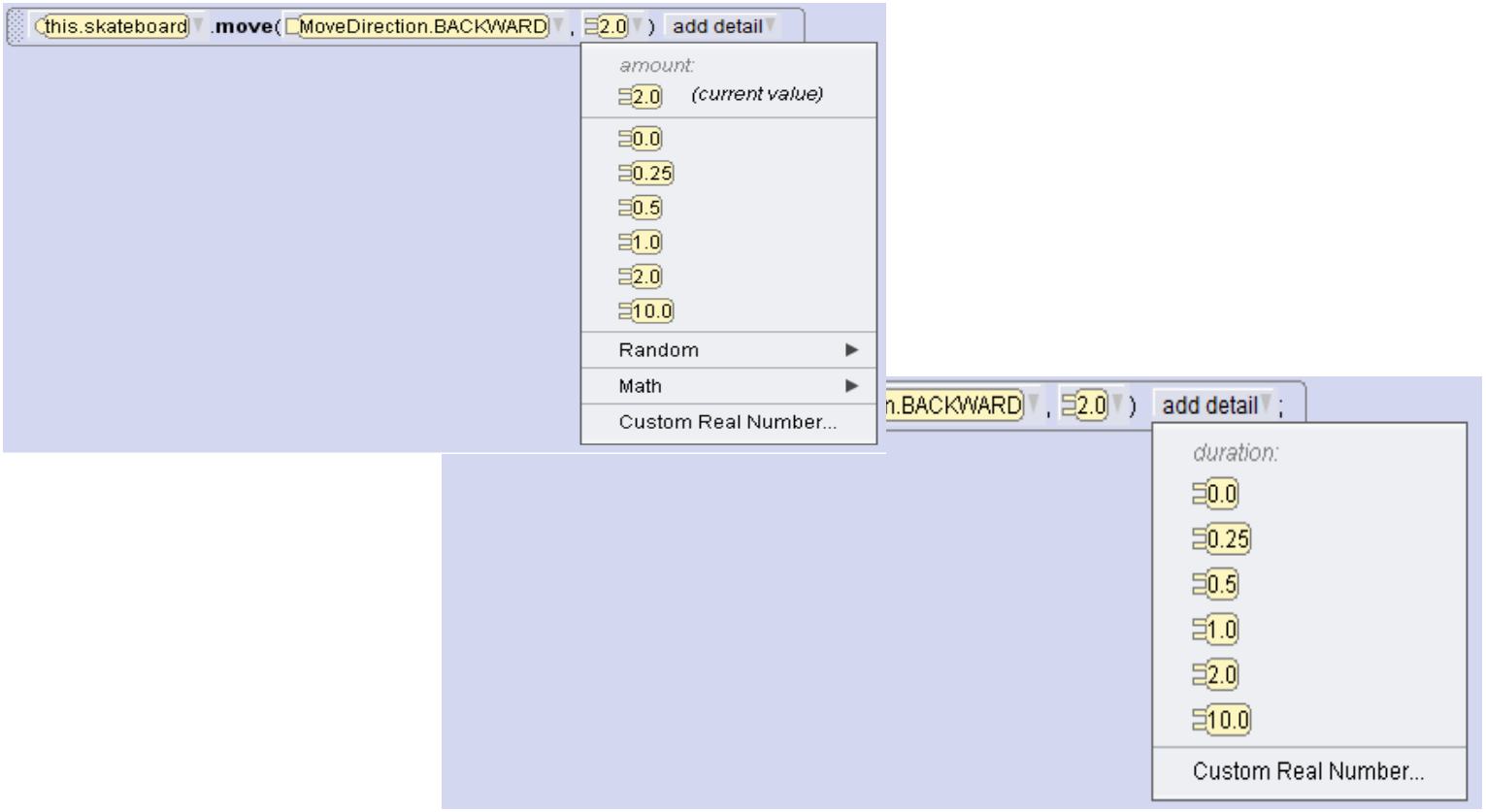
When a procedure tile is dropped into the code editor, its arguments may be changed, or further defined, to control the object's movement and timing.



A computer program requires arguments to tell it how to implement the procedure. Examples of Alice 3 arguments are: object, direction, direction amount, and time duration.

Editing Arguments (cont.)

Use the arrows next to the argument's value to display the menu and select a new value.

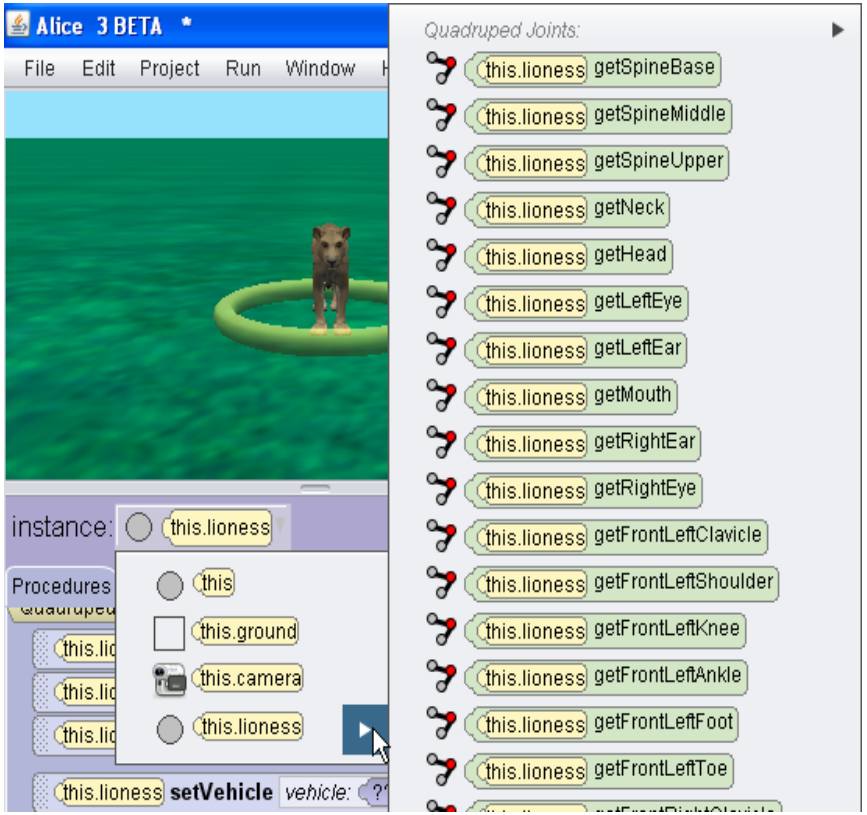


Editing Arguments (cont.)

In the Instance menu, you can select or change the object to program.

The Methods Panel will display all of the available options for the object.

You can also select the arrow next to the object's name to select a sub-part, if it has any.



Simultaneous Movements

Do In Order is the default sequential control statement in the code editor. This statement executes procedures in sequential order. For example:

- Object turns right, then
- Object turns left, then
- Object rolls right

However, sometimes objects (and their sub-parts) need to move together at the same time, such as for walking or sitting motions, which requires the Do Together control statement.

Simultaneous Movements (cont.)

To create the walking motion with a bi-pedal object, a series of procedures and Do Together control statements are required. Each bipedal object may walk a little differently and require different coding to walk properly.

Practice walking forward slowly. How do the subparts of each of your legs move as you walk forward?

You can document this in a textual storyboard.

Simultaneous Movements (cont.)

Textual Storyboard for Walking Motion

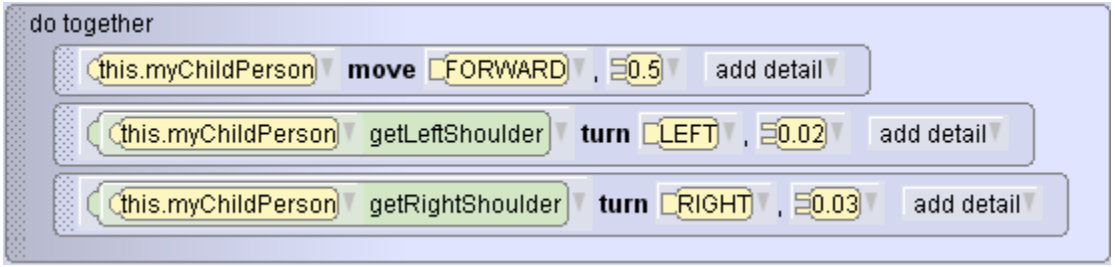
Left hip turns forward
Do Together
Whole body moves forward
Left hip turns backward
Right hip turns forward
Left shoulder turns left
Right shoulder turns right
Do Together
Whole body moves forward
Left hip turns forward
Right hip turns backward
Left shoulder turns right
Right shoulder turns left
...

Simultaneous Movements (cont.)

To program the walking motion for a person:

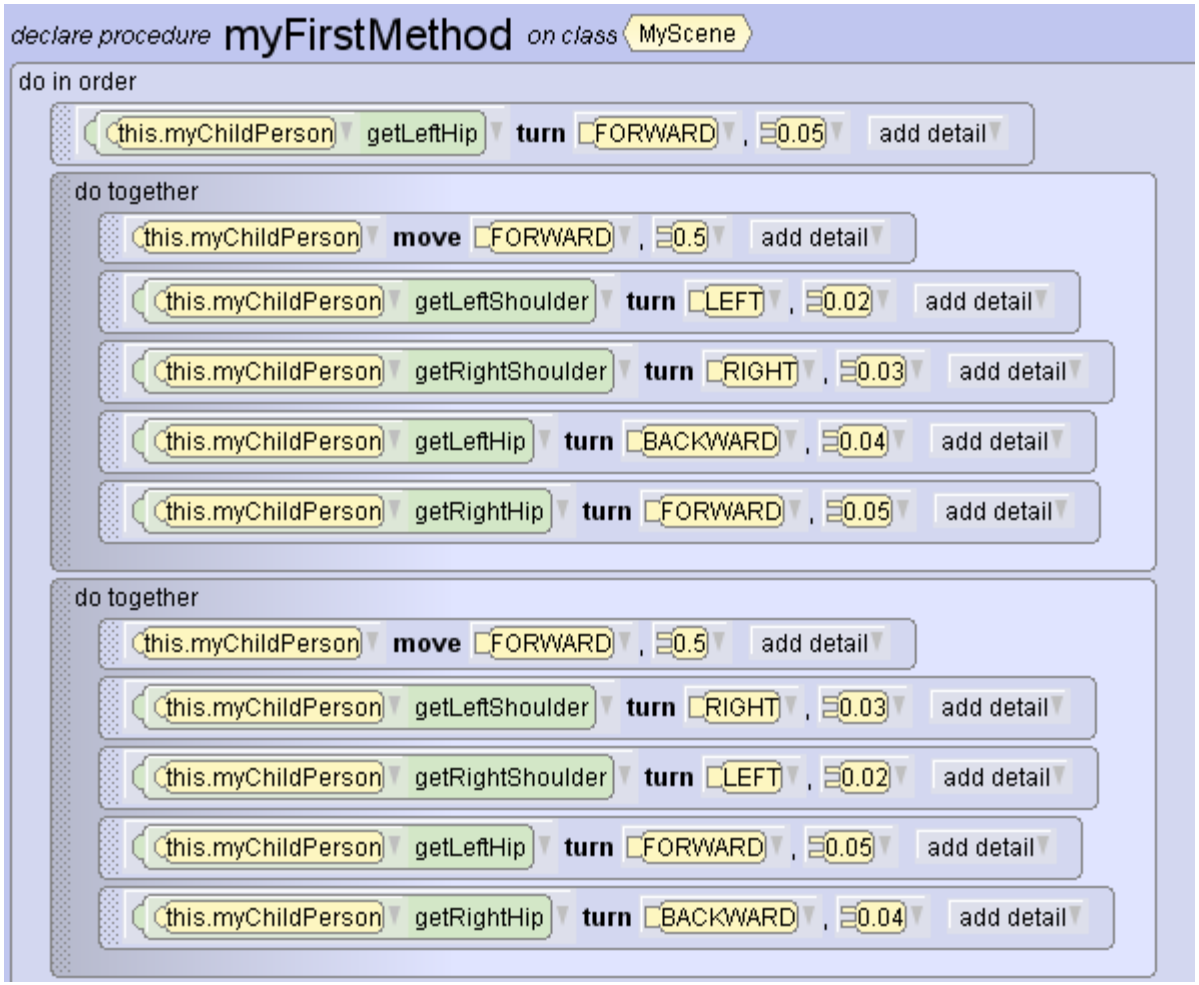
1. Drag the Do Together tile into the code editor.

2. Insert the procedures into the Do Together.



Simultaneous Movements (cont.)

Examine this code for a simple walking motion.



Vehicle Property

Another type of simultaneous movement is to have one object act as a vehicle of another. This means that the two objects are synchronized and move together.

Examples of this relationship:

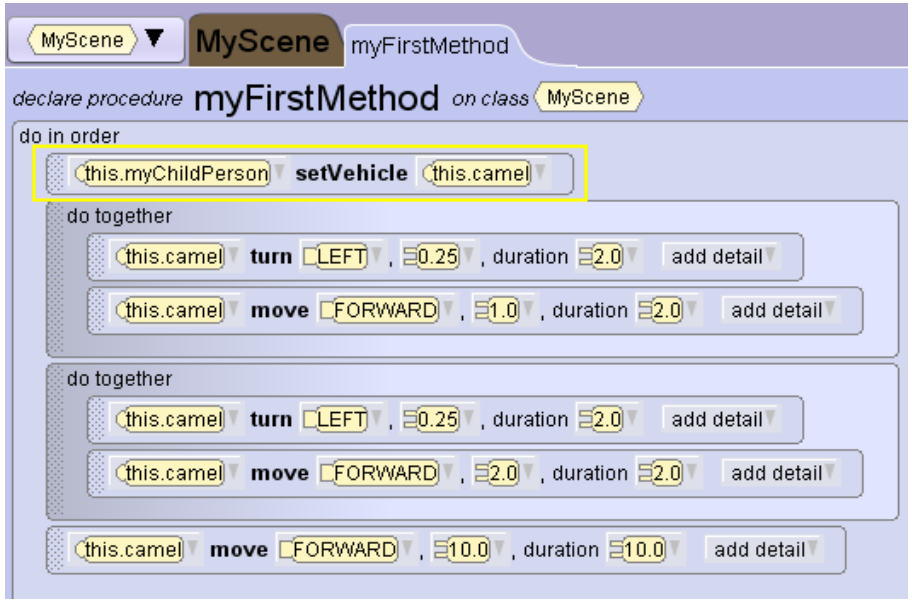
- A dog walks alongside a person
- A person rides a camel or horse
- A camera follows around a helicopter to shoot the scene from the helicopter's point of view

When one object is set as a vehicle of another using the setVehicle procedure, if you program the vehicle object to move, the rider object will automatically move with it.

Vehicle Property (cont.)

Examples

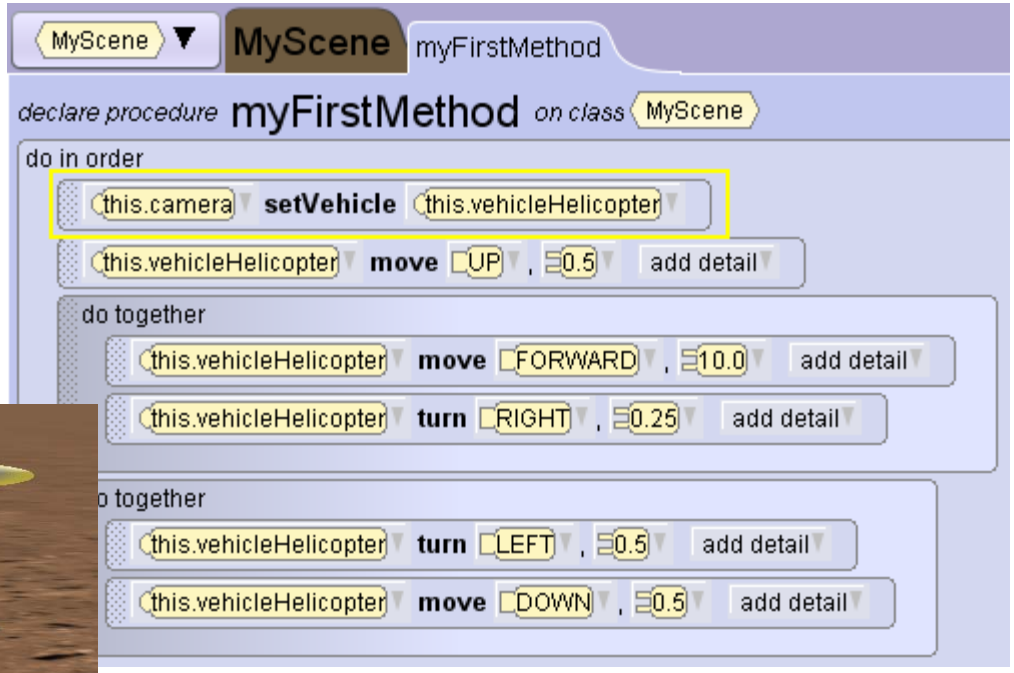
The child is positioned on the camel. Then, the camel is set as the vehicle of the child. When the camel is programmed to move, the child will stay on top and move with the camel.



Vehicle Property (cont.)

Examples (cont.)

The helicopter is set as the vehicle of the camera. When the helicopter moves around the scene, the camera films the scene from the helicopter's perspective.



Vehicle Property (cont.)

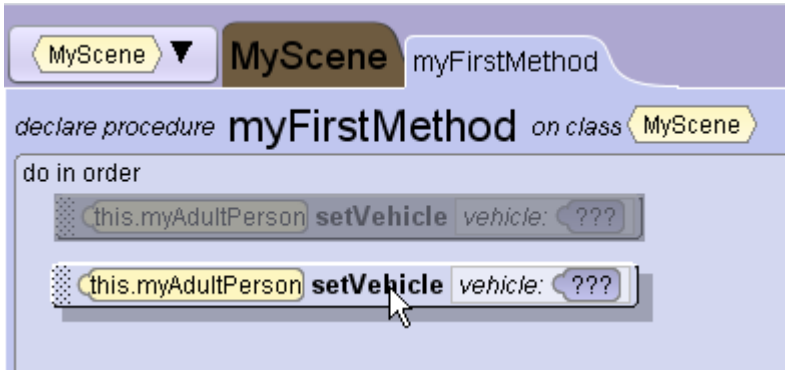
To assign an object as a vehicle of another object:

1. Determine the vehicle and the rider.

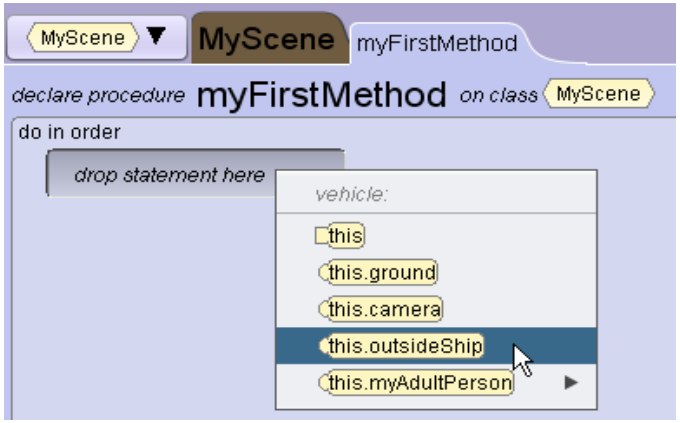
2. Select the rider object with your cursor.

Vehicle Property (cont.)

3. From Procedures tab in the Methods Panel, drag the setVehicle programming instruction into the code editor.

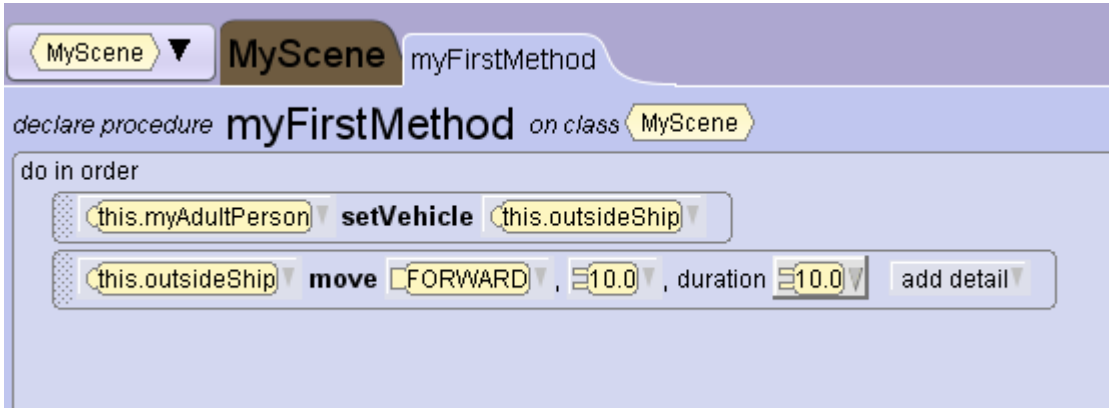


4. Select the vehicle object from the menu.



Vehicle Property (cont.)

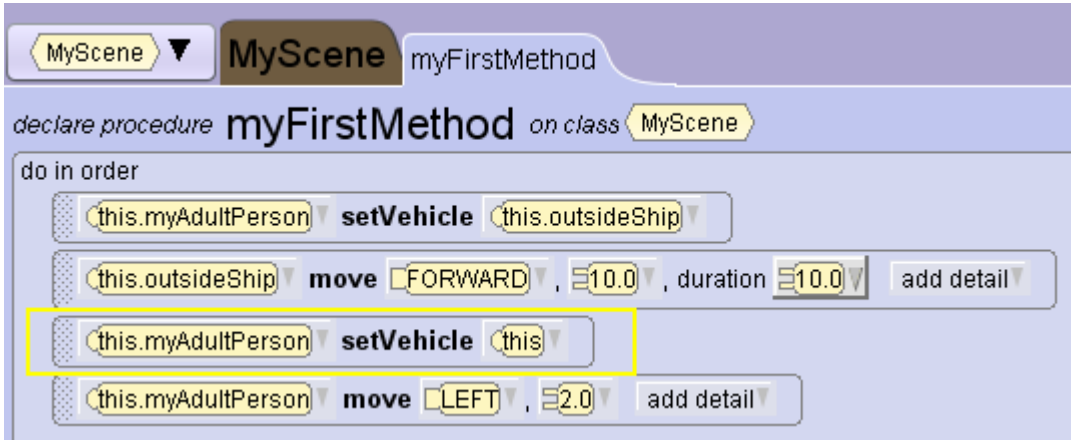
5. Insert the procedures to make the vehicle move. The rider will move along with the vehicle.



Vehicle Property (cont.)

If at some point during the animation you want the rider to no longer be connected with the vehicle:

1. Drag another setVehicle procedure into the code editor at the point the rider should get off the vehicle.
2. Set the vehicle to *this* (you are setting the vehicle of the rider back to the scene).
3. Continue programming your animation.



Functions

Programming an object's motion requires precision: very specific instructions to ensure the object acts exactly as you intend.

Precision may be achieved using trial-and-error. For example, you may continuously adjust the distance that the dog moves to the tree, and test the procedure over and over, until it moves and stands perfectly next to the tree.

However, there is an easier way to program the distance that an object moves. Alice 3 provides functions that can provide information about an object, such as its depth, the direction it is facing, or its distance to another object.

Functions (cont.)

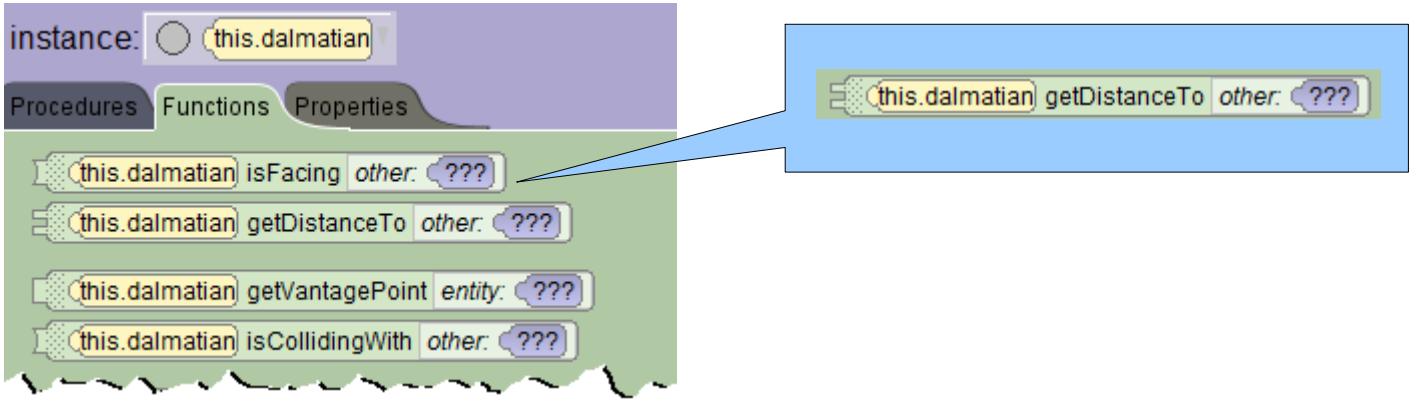
Functions answer questions about an object, such as its height, width, depth, and even its distance to another object.

Functions provide precise answers to questions, such as:

- What is the distance between the helicopter and the ground?
- What is the height of the cat?
- What is the width of the owl?

Functions (cont.)

Functions for an object can be found in the Methods Panel by selecting the Functions tab. View each object's functions in the code editor by selecting the object from the instance menu, then selecting the Functions tab.



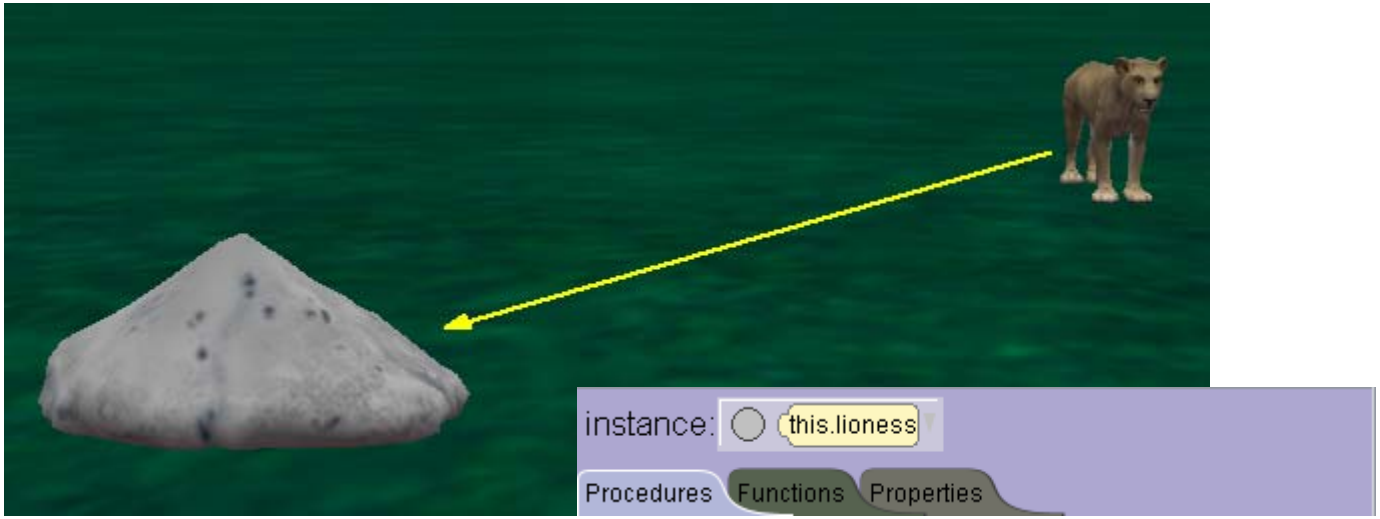
To use a function, select and drag a function tile onto an argument of an existing procedure in the code editor.

Functions (cont.)

For example, we want the lion to move directly to the rock without having to manually determine the distance between the lion and the rock.

Steps:

1. Identify the moving object and the target object. Select the moving object from the instance menu.

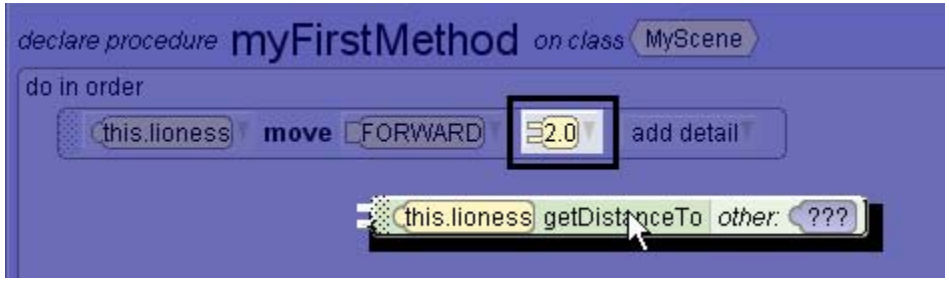


Functions (cont.)

Steps (cont.):

2. Drag the move procedure into the code editor.

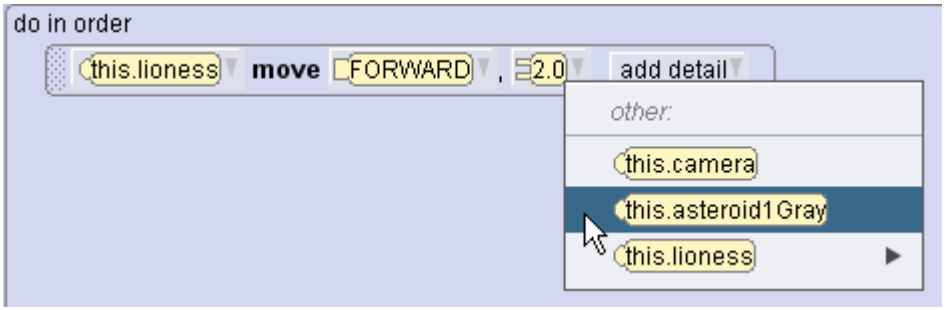
3. Select any placeholder arguments for direction and distance (the distance value will be changed in the next step).
4. From the Functions tab, drag the getDistanceTo tile onto the highlighted distance value.



Functions (cont.)

Steps (cont.):

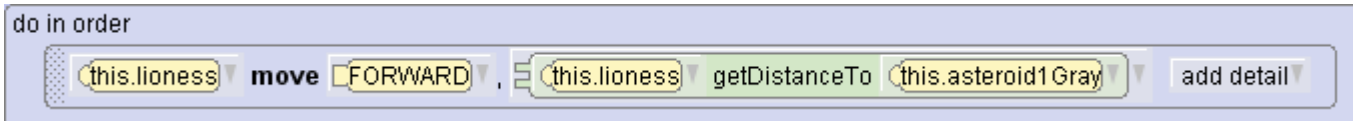
5. Select the target object. To verbalize this procedure you can say “the lioness will move forward to the rock the distance amount determined by the getDistance function.”



Functions (cont.)

Steps (cont.):

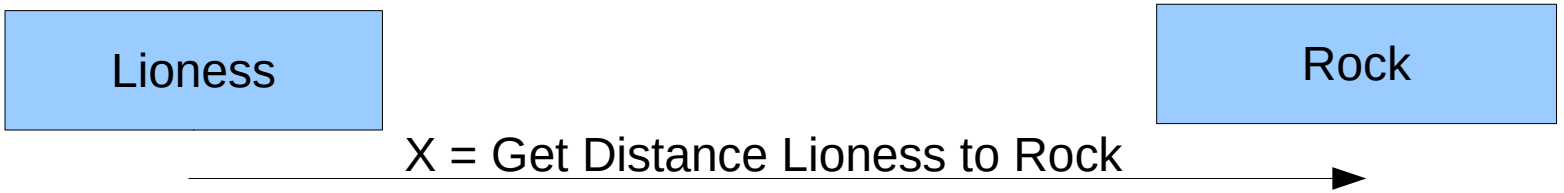
6. Click the Run button. Notice the lion moves to the center of the rock at run-time.



Functions (cont.)

The lion should land near the rock, but not in the center of it. To avoid the collision, you can use math operators to reduce the distance from the moving object to the target object.

To verbalize this example you can say “the lioness will move forward to the rock (Z) the distance amount determined by the getDistance function (X) minus the amount determined by calculating the width of the rock (Y) divided in half.”



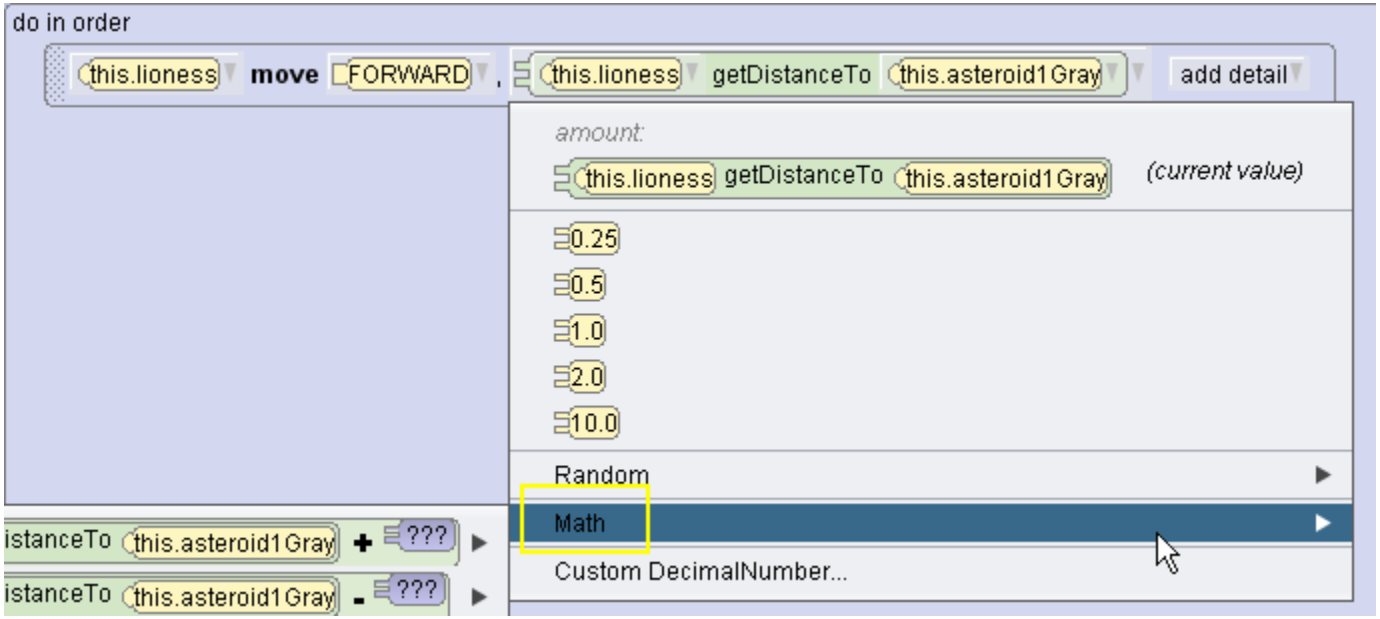
Z = X – (Y / 2)

Functions (cont.)

Steps:

1. Click the outer arrow next to the function argument.

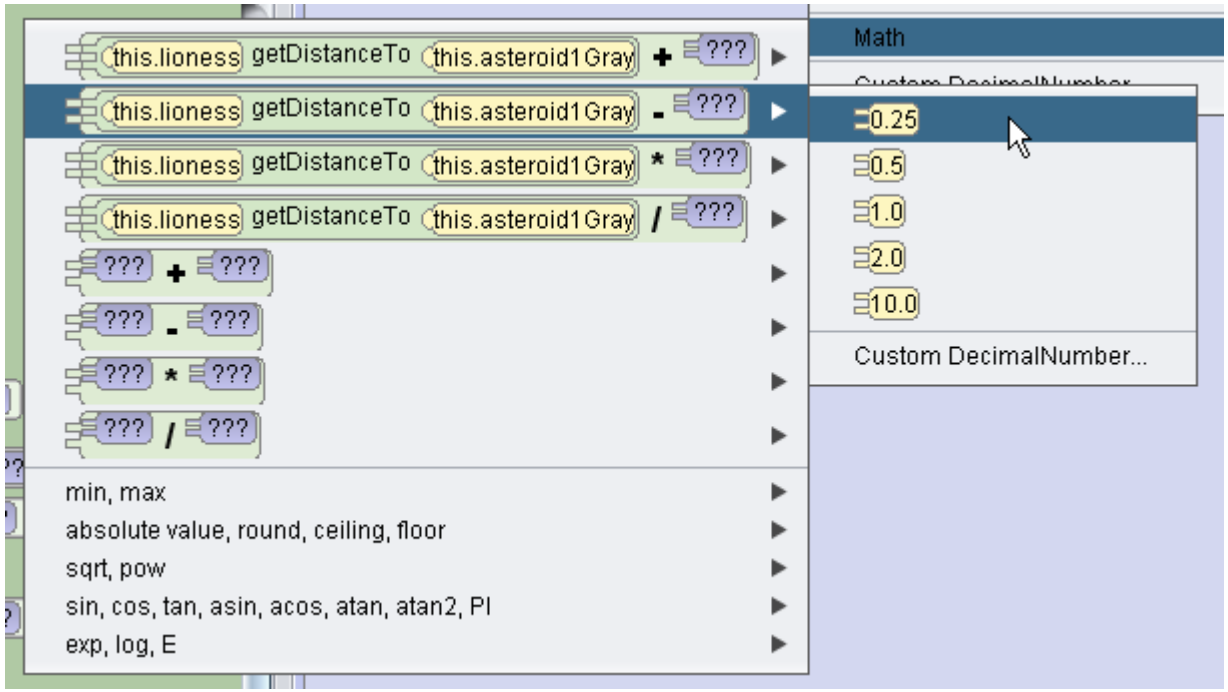
2. Select Math.



Functions (cont.)

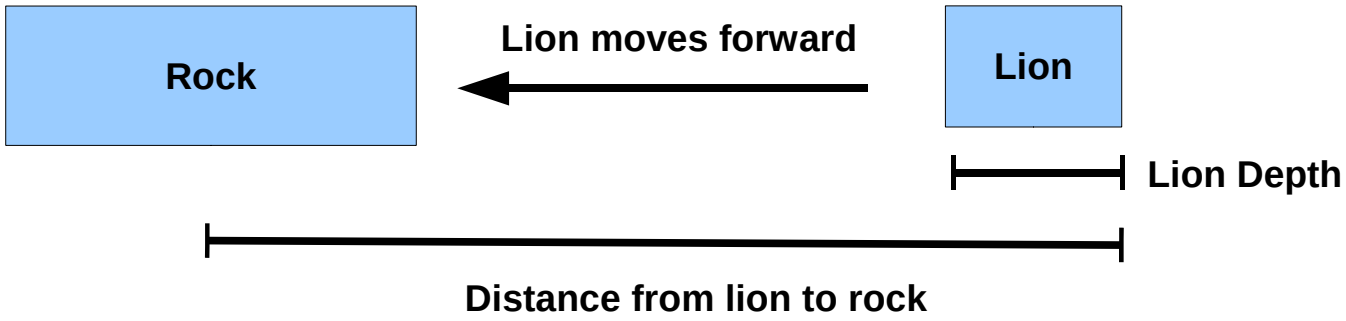
Steps (cont.):

3. Select the getDistanceTo subtraction option containing the placeholder ??? symbols.
4. Select the distance amount for the placeholder.
5. Run the animation to test the distance the object moves at run-time.



Functions (cont.)

A precise way to avoid a collision is to remove the length of the moving object from the function.

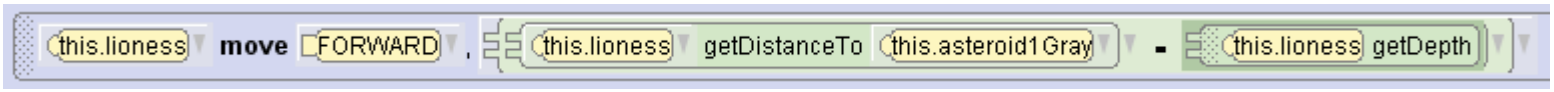


Functions (cont.)

To remove the length of the moving object from the function:

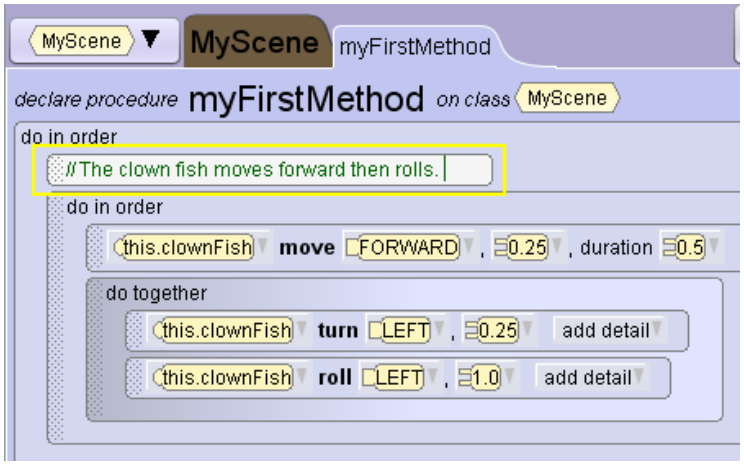
1. Drag the moving object's getDepth function onto the highlighted distance value.

2. Run the animation to test the complete programming statement. Adjust with additional math calculations if necessary.



Comments

Remember to include comments above each segment of programming statements.



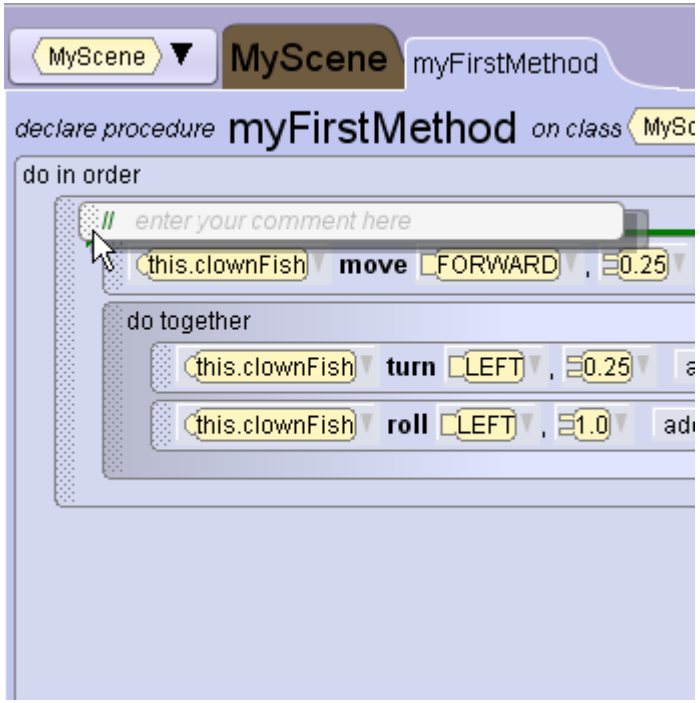
Comments:

- Help humans understand what your program does
- Describe the intentions of the programming instructions
- Do not affect the functionality or behavior of instances

Comments Steps Reviewed

To enter comments:

1. Drag and drop the comments tile above a code segment
2. Write comments that describe the sequence of actions in the code segment



In large programs it is not uncommon to create segment comments before creating programming instructions.

Terminology

Key terms used in this lesson included:

Collision

Comments

Functions

Object-orientation

Procedure arguments

Rotation

Simultaneous

Trial and error

Vehicle

Summary

In this lesson you learned how to:

- Define multiple control statements to control animation timing
- Create an animation that uses a control statement to control animation timing
- Recognize programming constructs to invoke simultaneous movement
- Use functions to control movement based on a return value
- Create programming comments

Practice

The exercises for this lesson cover the following topics:

- Writing programming comments
- Turning, rolling, and controlling actors in a scene
- Using math and functions to control movement