

In this lesson we will learn several tools and techniques to create some interesting movies.

## Functions

Functions are like procedures, except that procedures perform an action while functions return an answer.

Question: How far does an actor have to move to be next to something?

Answer:

The function `yeti.getDistanceTo(iceberg)` will return the distance from the yeti's pivot point to the iceberg's pivot point. If we move that distance the yeti will be in the center of the iceberg.

We need to move the yeti the distance to the iceberg - half the width of the iceberg and half the width of the yeti.:

```
Double distanceToMove = yeti.getDistanceTo( iceberg );
distanceToMove = distanceToMove-iceberg.getWidth()/2.0;
distanceToMove = distanceToMove-yeti.getWidth()/2.0;
```

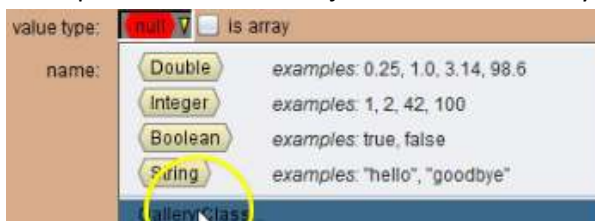
Once we get this working, we will move all of that code into a while block, then copy it to the clipboard.



Select Add Biped Function as shown above. Name the function **distanceNextTo** as shown.

Copy the code from the clipboard. Change "yeti" to "this" so that the function can work with other bipeds. At this point, the function will find the distance from any biped to the iceberg.

Add a parameter named object and select Gallery Class as the type:



Author: Janet E. Joy; Publisher: Zebra0.com

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/) Creative Commons Attribution-NonCommercial 4.0 International License

After selecting Gallery Class, you need to choose the highest item on the list that has the function getWidth that we are using in the code. You can see that the SModel class has the necessary function:

**class SModel***procedures*

- setVehicle
- setPaint
- setOpacity
- setWidth
- setHeight
- setDepth
- resize
- resizeWidth
- resizeHeight
- resizeDepth

*functions*

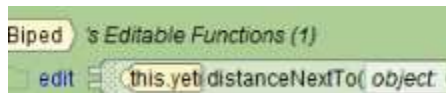
- getPaint
- getOpacity
- getWidth
- getHeight
- getDepth

Change each instance of "iceberg " to "object".

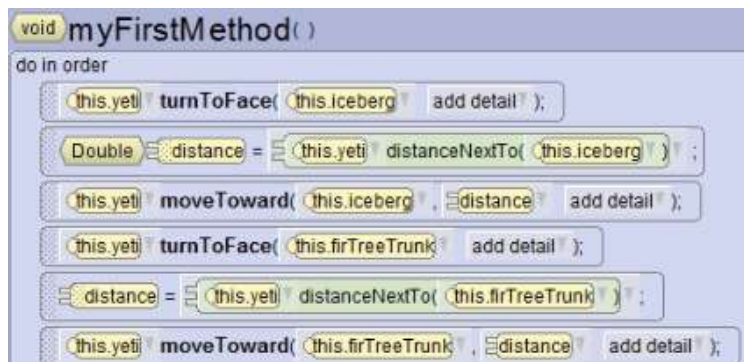
The last statement in a function is to return the answer:

```
public Double distanceNextTo( SModel object ) {
    Double distanceToMove = this.getDistanceTo( object );
    distanceToMove = distanceToMove-object.getWidth()/2.0;
    distanceToMove = distanceToMove-this.getWidth()/2.0;
    return distanceToMove;
}
```

The function will now be available for all bipeds.



We can use the function in our code using any object we want to move next to as the argument:



## Markers

Markers can be used to remember the position and orientation of an object. It is useful when you want to return an object to its original position after it has moved.

Once a marker has been set, it can be used by other objects.





To set a marker, place an object and orient it to the desired position. Select Add Object Marker. Name it a good descriptive name like nextToGate or inFrontOfBamboo.



Author: Janet E. Joy; Publisher: Zebra0.com

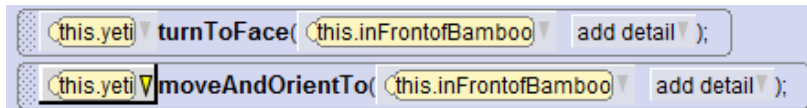
This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/) Creative Commons Attribution-NonCommercial 4.0 International License

When you select an object marker it will have handles and you can move it around just like any object.

You can move the selected object to a marker. Clicking  will move the yeti to the inFrontOfBamboo marker. Clicking  will move the marker to the yeti.

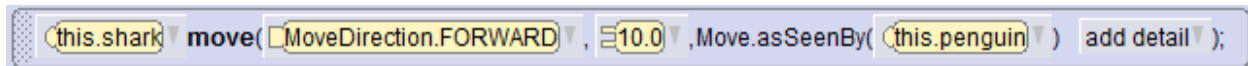


You can use the markers in your code and they will appear on the list of objects when you select commands such as turnToFace or moveTo.



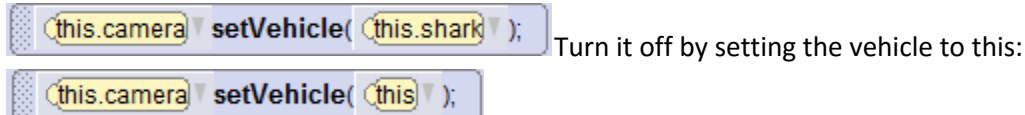
## As Seen By

The asSeenBy argument lets you see a movement from the perspective of another object. It is worth spending some time experimenting with this to see how the movie looks from different perspectives.



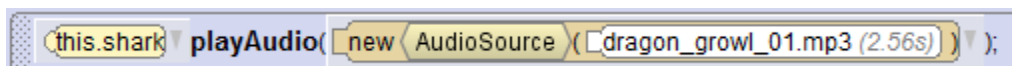
## Vehicles for the Camera

If you assign a vehicle to the camera, you can give the feeling of riding along with the object. You can assign a vehicle to the camera in either design view or in code:



## Sounds

Before you can play a sound, you need to import it to the project. Select project resource manager and import an audio. There are many sounds in Alice, but you can import sounds from other sources. You can even create your own sounds. [Audacity](#)® is free, open source, cross-platform audio software for multi-track recording and editing. There are also many websites where you can download sounds for free. Search for free sound effects.



We will discuss playing background sounds in the lesson on events.

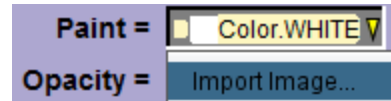


Author: Janet E. Joy; Publisher: Zebra0.com

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) Creative Commons Attribution-NonCommercial 4.0 International License

## Billboards

Billboards are found in Shapes/Text in the gallery. If you position a billboard in the background, then add an image as the paint color, there is no limit to the scenes you can create. You could find a picture of the Eiffel Tower and add it to a billboard as the paint color. You could also find a picture of something that you could not find in Alice. If you have a transparent gif, set the opacity to 0.9 to make the picture transparent. Remember that when you use images, they are not 3D models and they cannot turn and behave like 3D objects.



## Scene Changes using Opacity

One of the ways to create scene changes is to change the ground, then make some objects visible (opacity =1.0) and other objects invisible (opacity =0.0) You can use an array if there are several items. You can set the duration to 0 so that this all happens immediately. Here is a simple change from the desert to the snow:

```
public void myFirstMethod() {
    this.bunny.say( "It's too hot here in the desert." );
    this.bunny.say( "I'm going someplace cool." );
    this.setAtmosphereColor( Color.CYAN, SetAtmosphereColor.duration( 0.0 ) );
    this.pricklyPearCactus.setOpacity( 0.0, SetOpacity.duration( 0.0 ) );
    this.iceFloe.setOpacity( 1.0, SetOpacity.duration( 0.0 ) );
    this.bunny.moveTo( this.bunnyOnIce, MoveTo.duration( 0.0 ) );
    this.ground.setPaint( SurfaceAppearance.SNOW, SetPaint.duration( 0.0 ) );
    this.bunny.say( "Now I'm too cold." );
}
```

You can also use a billboard in the front of the scene and make it visible during the transition, then invisible when the transition is finished.

## Scene Changes using Transitions

Instead of having the scene change abruptly, you can also put a billboard in front of the scene and have it fade out gradually. Experiment to see what effects you can create and how to achieve them.

## Scene Changes using Camera Markers

One way to create scene changes is to create the Alice world with one scene on the left and a different scene on the right. Position the camera so the you only see the left side, then create a camera marker. Do the same for the scene on the right. In the movie you can move the camera to the marker to create a scene change, or you can combine that with any of the techniques above for scene changes.



Author: Janet E. Joy; Publisher: Zebra0.com

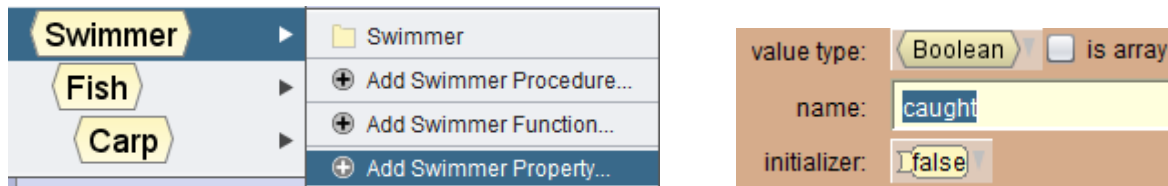
This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/) Creative Commons Attribution-NonCommercial 4.0 International License

## Scene Changes using Multiple Characters

Sometimes it is easier to have multiple versions of an actor. For instance, to tell the story of Cinderella there is one Cinderella dressed in rags and another dressed in a gown. Of course, you only see one of them at a time! This can be done either by making one invisible (opacity = 0.0) or by changing the camera angle so that only one is on camera at a time.

## Properties of a Class

A class property can be a way to solve several problems. Let's suppose you want to keep track of how many fish the shark caught. Adding an integer **fishCaught** to the shark will let you access the same variable in different procedures. You may want to add a Boolean property **caught** to the fish class so that you can keep track of which fish have been caught. There are countless ways that a class property can help you to code a complex situation.



After you have added a property, you will see a procedure to **set** the property:

`this.carp.setCaught( caught: ??? )`, and a function to **get** the property: `this.carp.getCaught()`.

You should experiment with these techniques to create an interesting project.



Author: Janet E. Joy; Publisher: Zebra0.com

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/) Creative Commons Attribution-NonCommercial 4.0 International License