Alice 3.0

INTRODUCTION

Welcome to Alice 3.0! In this section of SuperCamp we will build animations (and maybe even a Game or two) using this very versatile environment. You can download it yourself at www.alice.org at home as well.

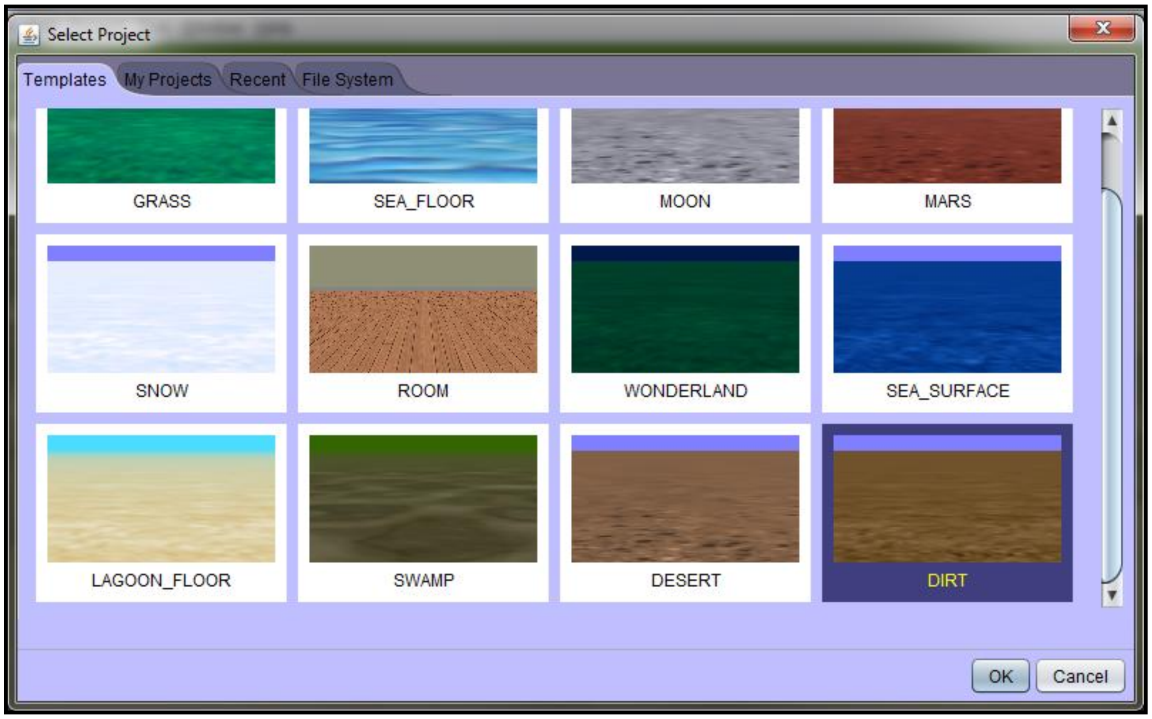
As you become more proficient at programming, you can transition from using the Alice point and click language to using Java and the NetBeans environment. Some of the labs contained in this booklet mention using this environment. We will skip these parts for now, but you may want to come back to them at home later. Remember yo will be able to down load all of the manuals we use at camp from our website at robocamp.cse.unt.edu.

Manipulating the Alice Environment

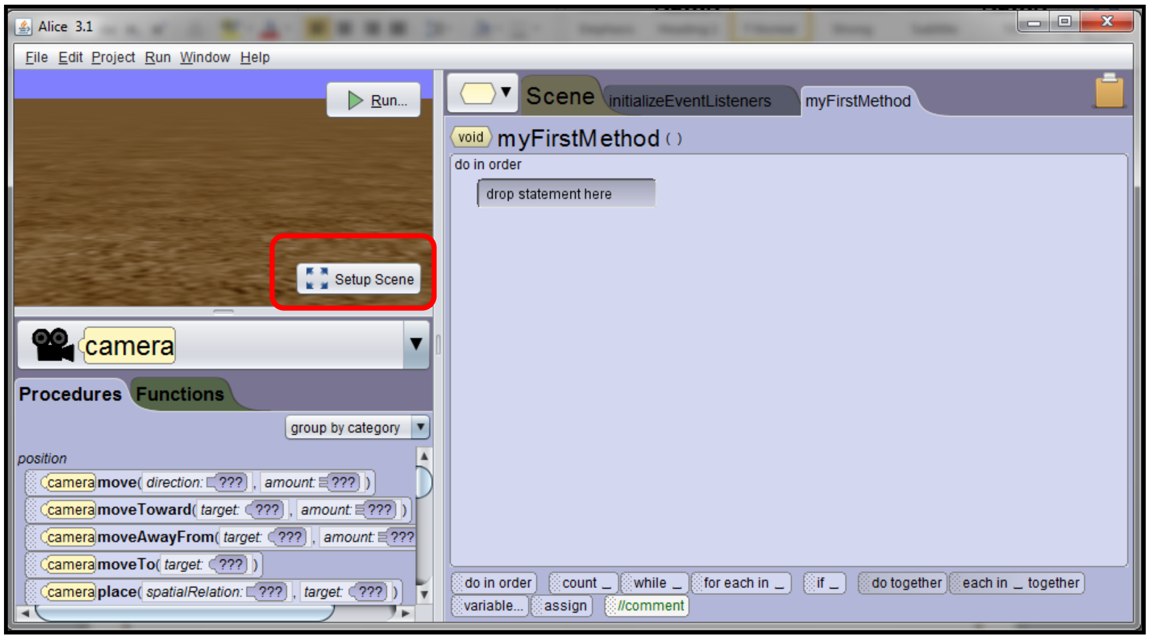
1. Open up Alice3. You will need to find the installed Alice3 folder and double click on the Alice3.bat batch file (alice3.sh for the Mac). A black window with command line will appear on your screen and should remain open (can be minimized) while Alice3 software is running.

2. We are going to add a few objects to the Alice environment so that we can manipulate the objects and scene to get a better idea of how Alice works.

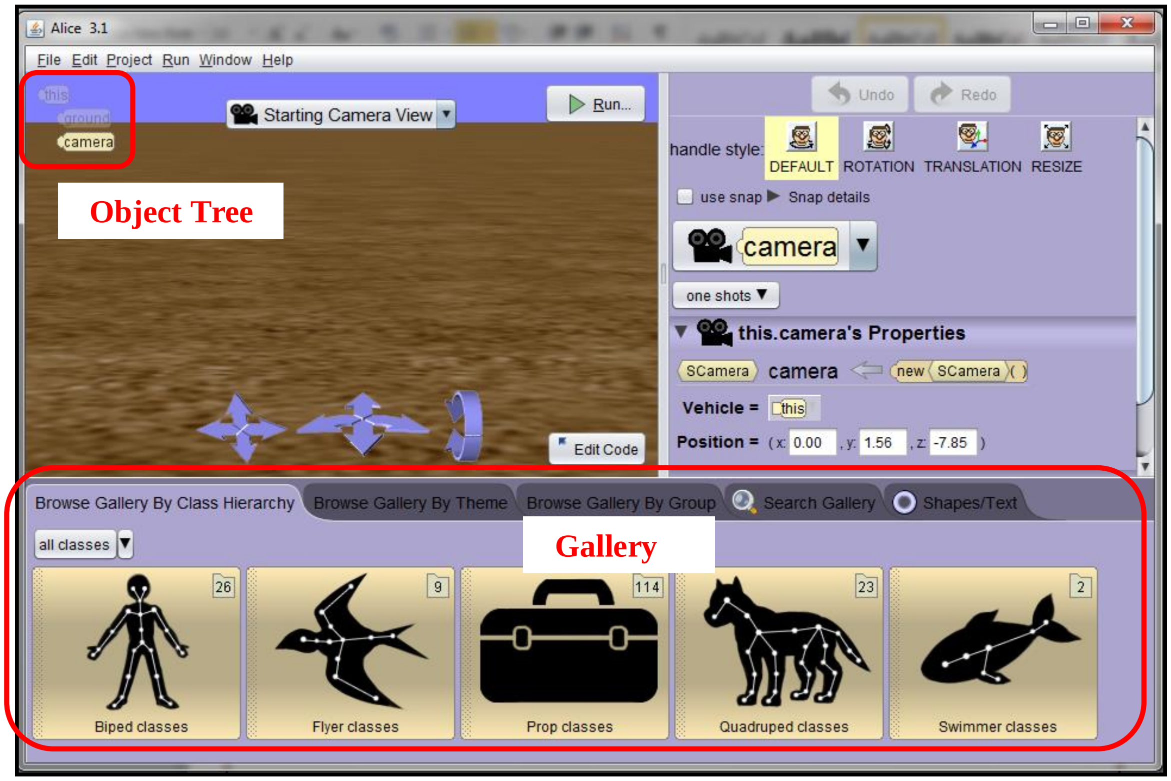
3. When you open the Alice program, you will be prompted to select a template. You also have the choice of selecting an old project. We will use this option at the end of this exercise. Make sure that the **Templates** tab is selected. Scroll down and select **Dirt**. Click **OK**.



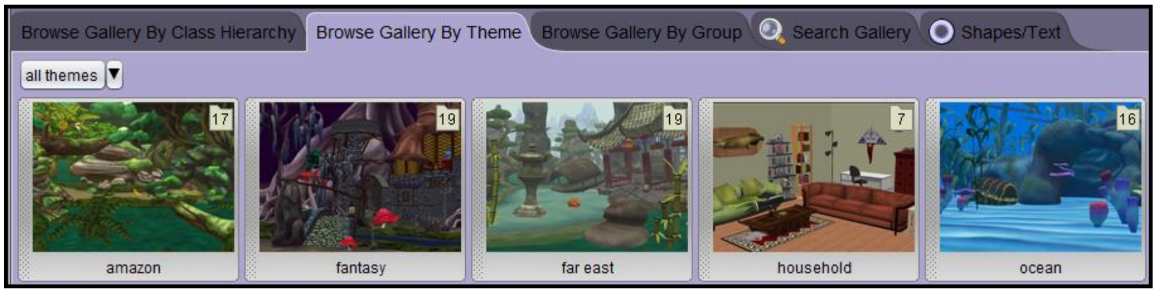
4. Your screen should look similar to the following. Before we can write any code, we need to add some objects to our scene. You will need to click on **Setup Scene** to add objects.

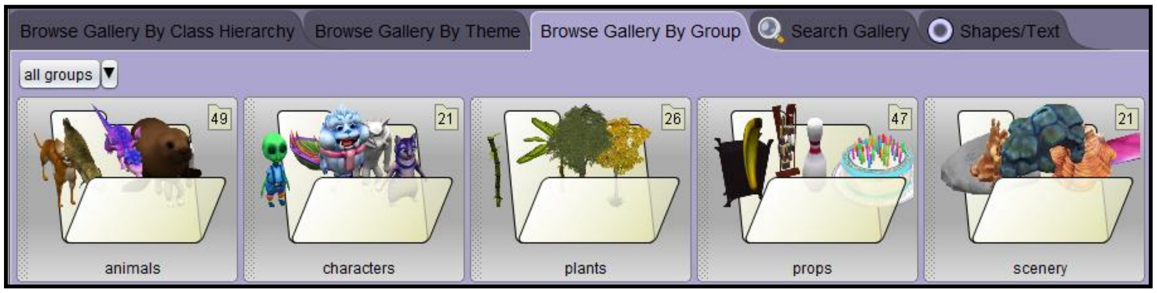

Alice 3.1
File Edit Project Run Window Help
Run...
Setup Scene
camera
Procedures Functions
group by category
position
camera move(direction: ??? , amount: ???)
camera moveToward(target: ??? , amount: ???)
camera moveAwayFrom(target: ??? , amount: ???)
camera moveTo(target: ???)
camera place(spatialRelation: ??? , target: ???)
Scene initializeEventListeners myFirstMethod
void myFirstMethod ( )
do in order
drop statement here
do in order count while for each in if do together each in together
variable... assign //comment

5. The scene setup area will look similar to the following. You currently have 2 objects in this scene: the camera and ground. You can see the objects in your scene by looking at the **object tree**. You can add objects to your scene by using the **gallery** options.

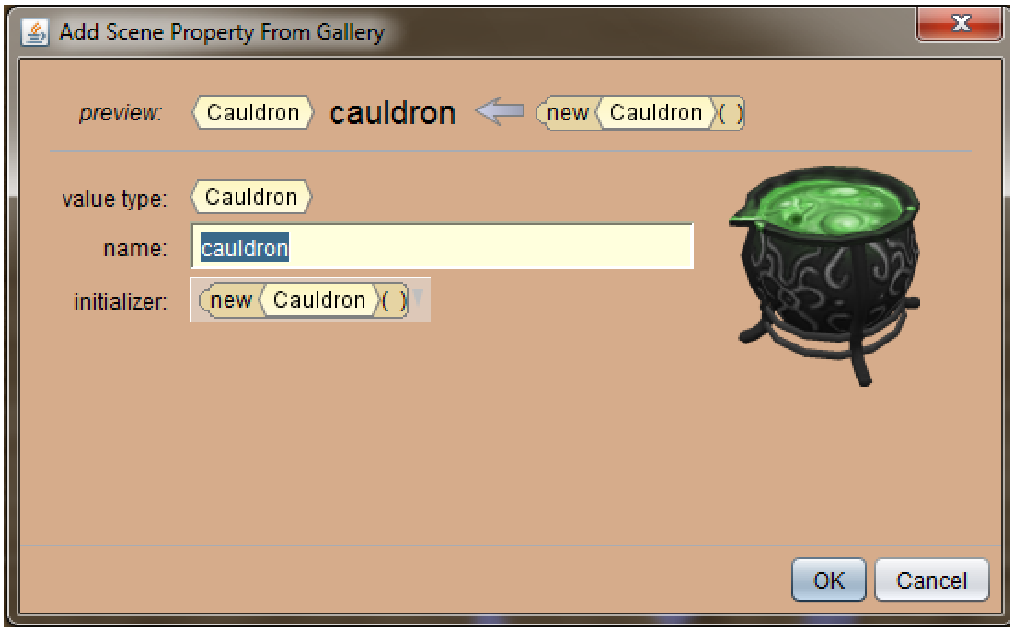

Alice 3.1
File Edit Project Run Window Help
this
ground
camera
Object Tree
Starting Camera View
Run...
Edit Code
Undo Redo
handle style: DEFAULT ROTATION TRANSLATION RESIZE
use snap Snap details
camera
one shots
this.camera's Properties
SCamera camera new SCamera ( )
Vehicle = this
Position = (x: 0.00 , y: 1.56 , z: -7.85)
Browse Gallery By Class Hierarchy Browse Gallery By Theme Browse Gallery By Group Search Gallery Shapes/Text
all classes
Gallery
26 Biped classes
9 Flyer classes
114 Prop classes
23 Quadruped classes
2 Swimmer classes

6. There are several choices for selecting objects from the gallery. Please take a moment to explore the possibilities.





7. Please click back on the **Browse Gallery by Class Hierarchy**. Then select **Prop Classes** and scroll until you find **Cauldron** (they are in alphabetical order). We are going to add this to our scene. Go ahead and click on it. You should be prompted to give this new object a name. We are going to leave the default name **cauldron** and click **OK**. Please leave the default names for this exercise.


Add Scene Property From Gallery

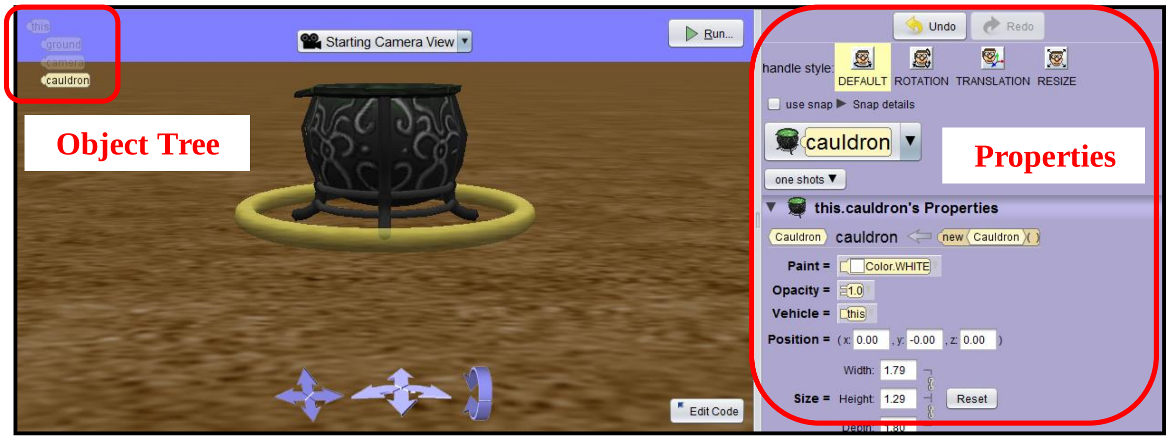
preview: Cauldron cauldron ⇐ new Cauldron ( )

value type: Cauldron

name: cauldron

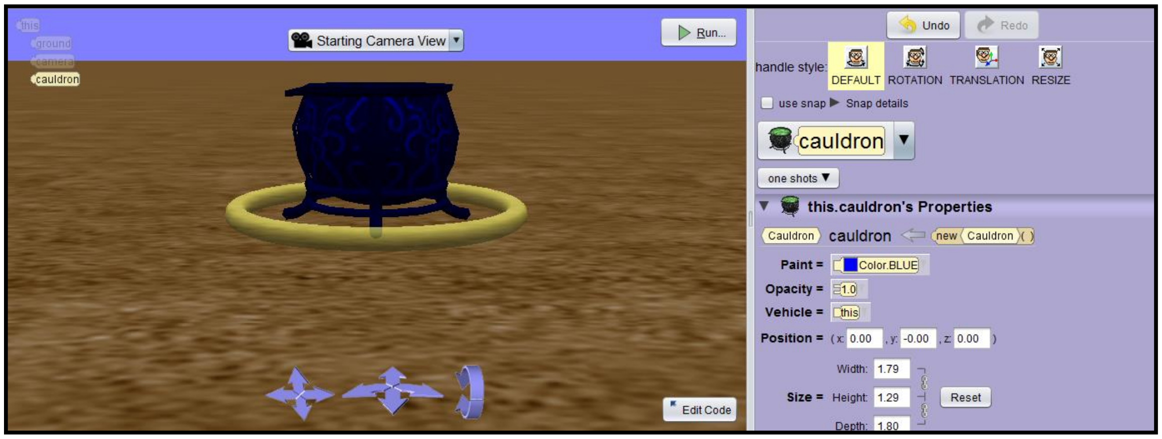
initializer: new Cauldron ( )

8. You will notice that the cauldron appears in the center of the scene by default. It is also added to your object tree. The properties for the object are listed on the right pane as shown below. You can change the location, orientation, color, opacity, size, etc.



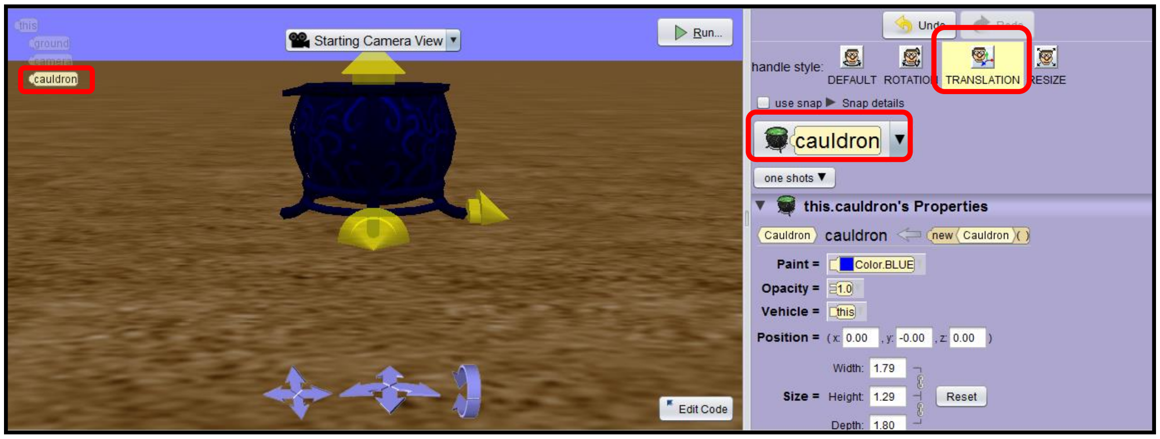
this.cauldron's Properties
Cauldron cauldron ⇐ new Cauldron ( )
Paint = Color.WHITE
Opacity = 1.0
Vehicle = this
Position = (x: 0.00, y: -0.00, z: 0.00)
Size = Width: 1.79 Height: 1.29 Depth: 1.80

9. Let’s change the color of the cauldron to blue. Click the drop down next to **Paint** and select **Color.BLUE**. You cauldron should change color.

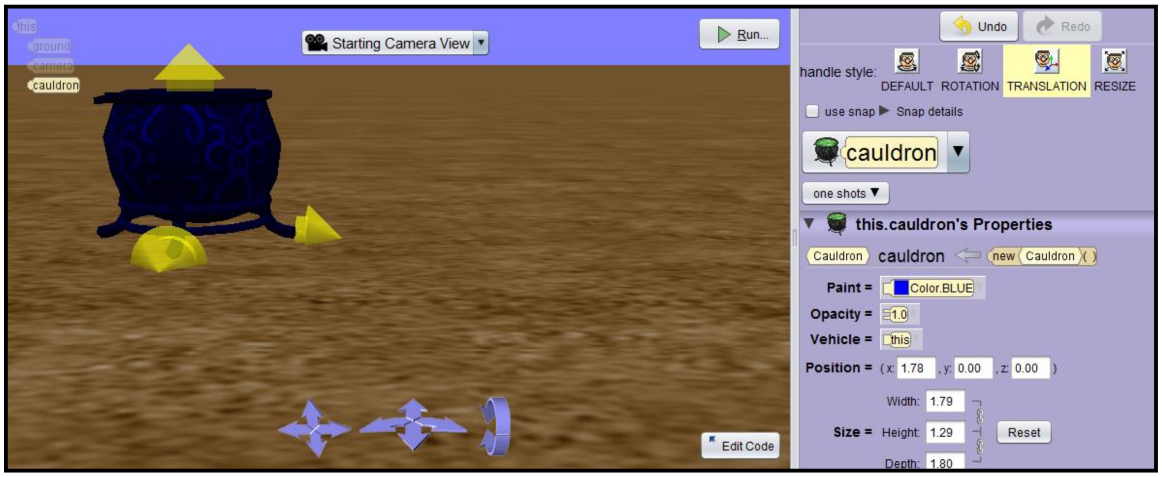


this.cauldron's Properties
Cauldron cauldron ⇐ new Cauldron ( )
Paint = Color.BLUE
Opacity = 1.0
Vehicle = this
Position = (x: 0.00, y: -0.00, z: 0.00)
Size = Width: 1.79 Height: 1.29 Depth: 1.80

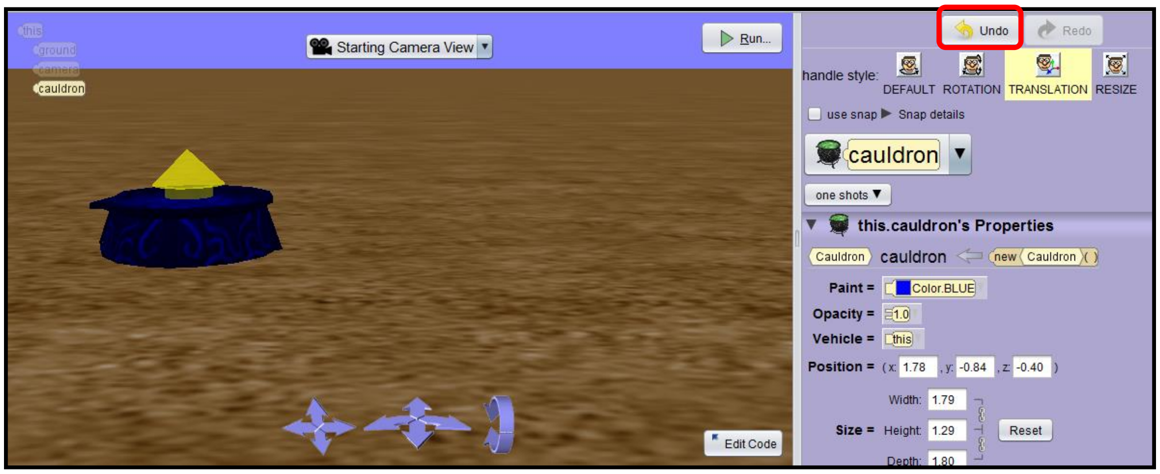
10. Next, we want to move the cauldron to the left side of the scene. The ring around the cauldron indicates that you can rotate the cauldron 360 degrees (this is the default option). I do not want to rotate the cauldron since it would look the same all the way around. I want to move it. To move an object, make sure that the object is selected from the object tree or from the drop down on the properties pane as shown below and select **Translation**.



11. You will notice that the handle style changes to arrows. The arrow on top of the cauldron will move the cauldron up and down if you hold down your left mouse button on the arrow and drag up or down. The arrow to the right of the cauldron, will move the cauldron to the right or left. The front arrow will move the cauldron forwards and backwards. Try moving the cauldron to the left and forward.



12. If you move the cauldron up or down, it will be off the ground. Go ahead and try it. You don't have to worry about messing up the environment because you can undo. If your cauldron is in the ground or floating in the air, click the **undo** button as shown below.

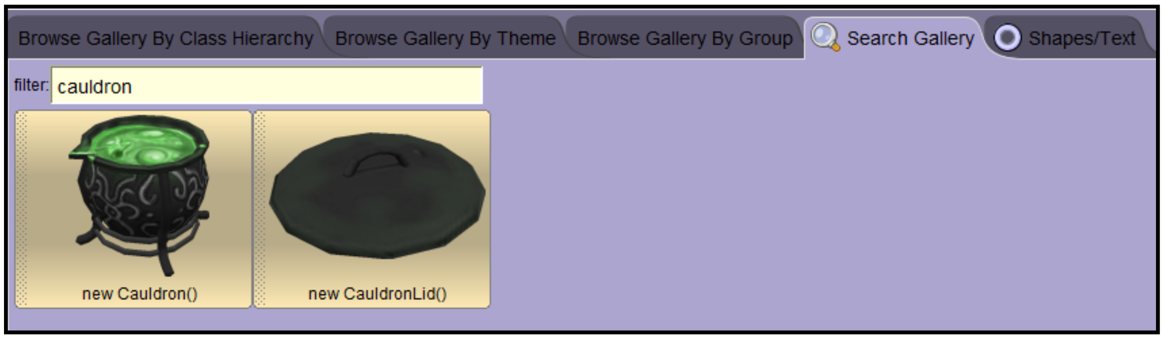


Cauldron cauldron ⇐ new Cauldron ( )

Paint = Color.BLUE
Opacity = 1.0
Vehicle = this
Position = (x: 1.78 , y: -0.84 , z: -0.40)
Size = Width: 1.79 Height: 1.29 Depth: 1.80 Reset

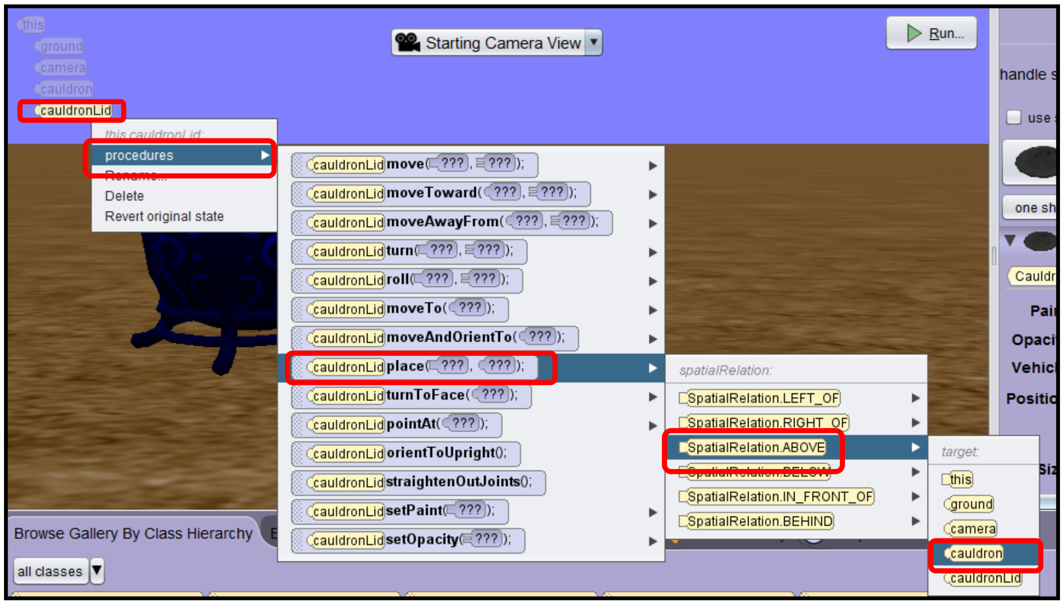
13. Do not resize the cauldron. We are going to add the cauldron lid and if we resize the cauldon, it won't fit and we will have to resize the cauldron lid to fit. We will practice with rotating and resizing in the next exercise.

14. Next, let's add the cauldron lid. See if you can find it. You can try using the search since we know the object that we want to add. Click on the CauldronLid to add a cauldron lid to your scene. Leave the default name for the object.

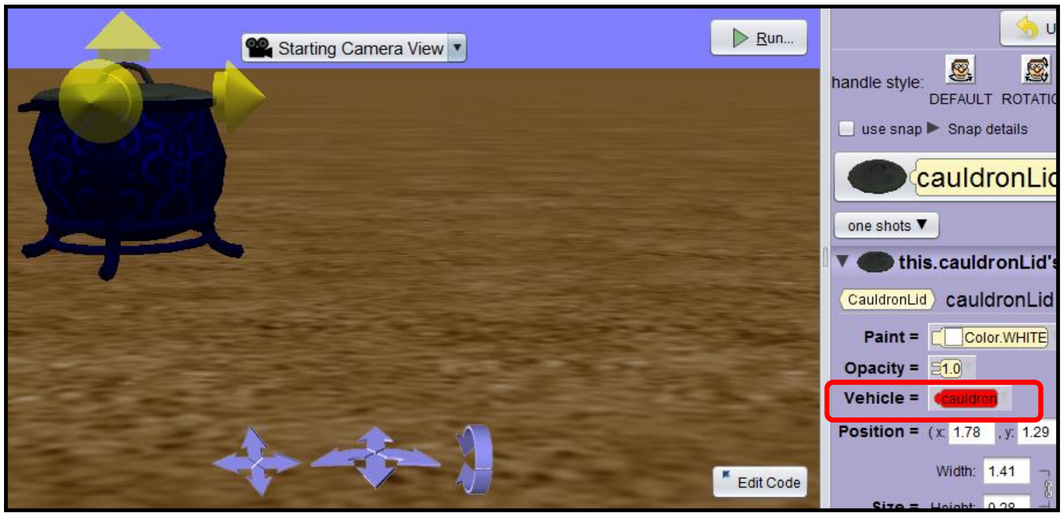


filter: cauldron

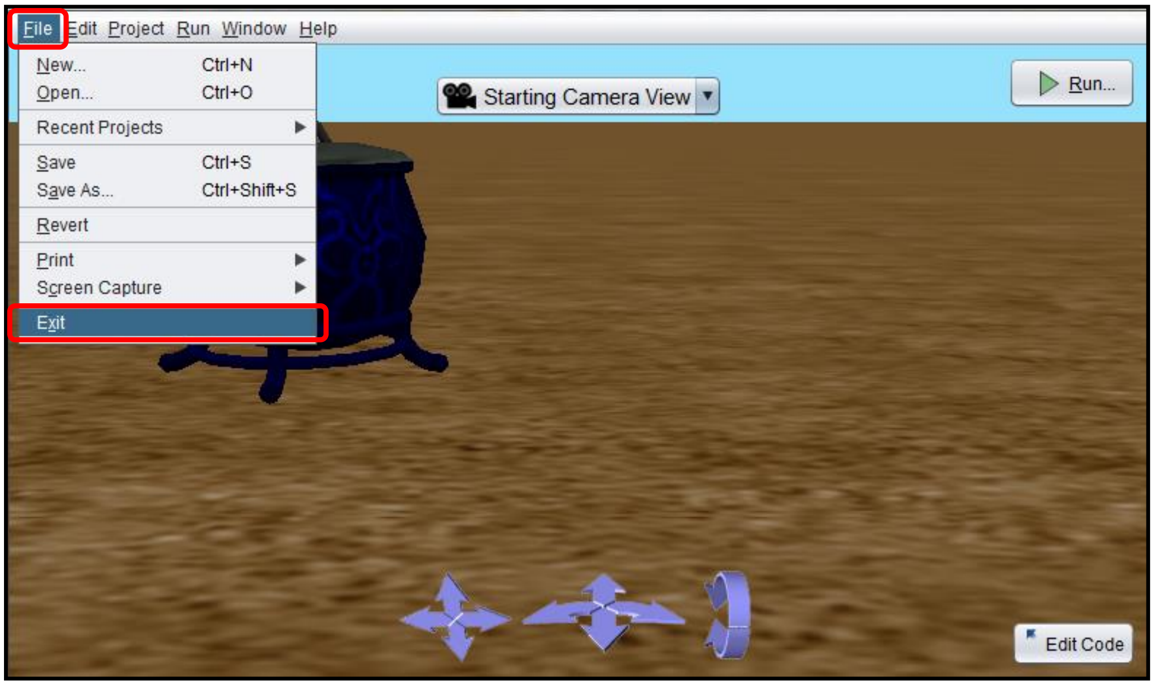
15. The cauldron lid is added to our scene, but it is on the ground. We could move the cauldron up, to the left, and forward to get it on top of the cauldron, but there is an easier way. There is a One Shot drop down that will allow us to move the cauldron lid onto the cauldron without having to move it ourselves. Make sure the cauldron lid is selected and right click on the cauldron lid from the object tree. Choose **procedures**, **place**, **above**, and select **cauldron**. Be careful with the cascading menus when you are selecting options. You can think of procedures as actions.


this
ground
camera
cauldron
cauldronLid
Starting Camera View
Run...
this.cauldronLid:
procedures
Rename...
Delete
Revert original state
cauldronLid move(???, ???);
cauldronLid moveToward(???, ???);
cauldronLid moveAwayFrom(???, ???);
cauldronLid turn(???, ???);
cauldronLid roll(???, ???);
cauldronLid moveTo(???);
cauldronLid moveAndOrientTo(???);
cauldronLid place(???, ???);
cauldronLid turnToFace(???);
cauldronLid pointAt(???);
cauldronLid orientToUpright();
cauldronLid straightenOutJoints();
cauldronLid setPaint(???);
cauldronLid setOpacity(???);
spatialRelation:
SpatialRelation.LEFT_OF
SpatialRelation.RIGHT_OF
SpatialRelation.ABOVE
SpatialRelation.BELOW
SpatialRelation.IN_FRONT_OF
SpatialRelation.BEHIND
target:
this
ground
camera
cauldron
cauldronLid
Browse Gallery By Class Hierarchy
all classes

16. The cauldron lid should now be on top of the cauldron, but if we move the cauldron the lid will not move with it. To make the lid move with the cauldron, we need to change the vehicle property of the lid to the cauldron. Make sure that the cauldron lid is selected and change the **vehicle property** to **cauldron** as shown below. Try moving the cauldron, does the lid move with it?

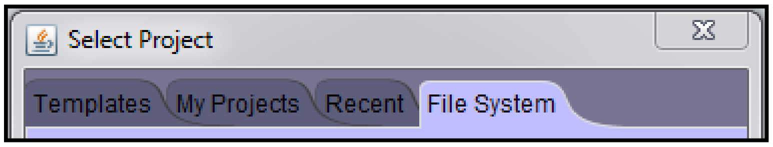


17. We are going to save our project and exit Alice. To save the project, you should click on **File**, **Save As**. Name the project **PracticeWithAlice**. Please get in the habit of capitalizing the first letter of every word in your filename and do not use space when naming your files. Click **File, Exit** to close out of the Alice program.

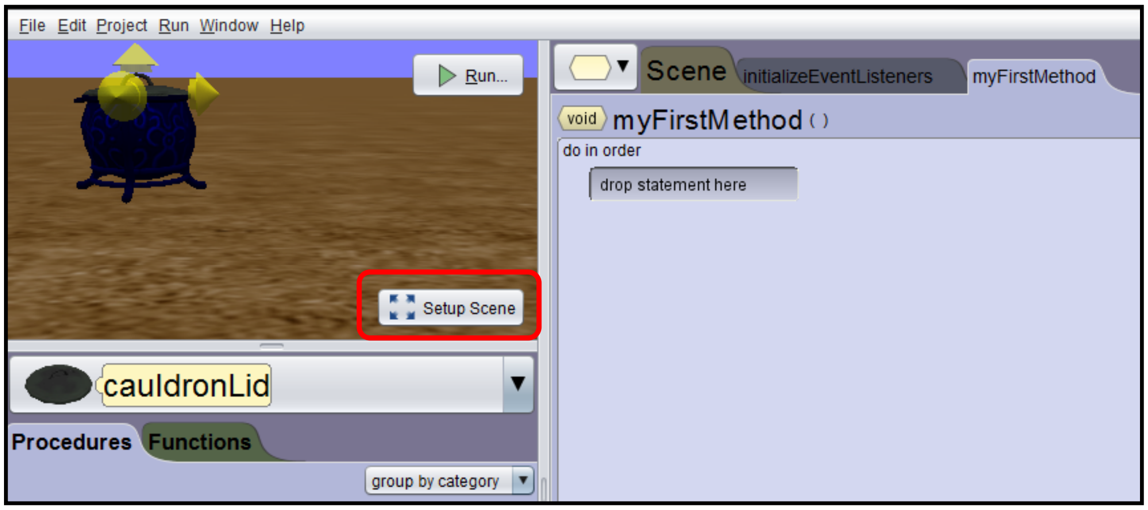


1. Open up Alice3. You will need to find the installed Alice3 folder and double click on the Alice3.bat batch file (alice3.sh for the Mac). A black window with command line will appear on your screen and should remain open (can be minimized) while Alice3 software is running.

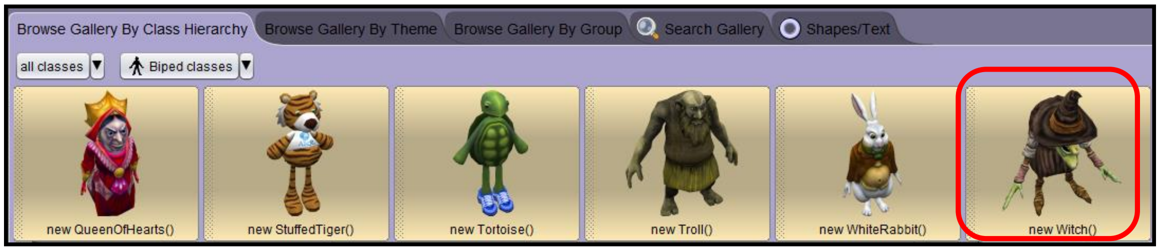
2. Click on the **File System** tab and choose **browse…** You will need to locate your **PracticeWithAlice** file and click **OK**. *(Note: Alice files have an .a3p extension. If you double click on this file, it will not open in Alice. Unless you create a file association, you will need to open all of your Alice projects from the Alice software.)*



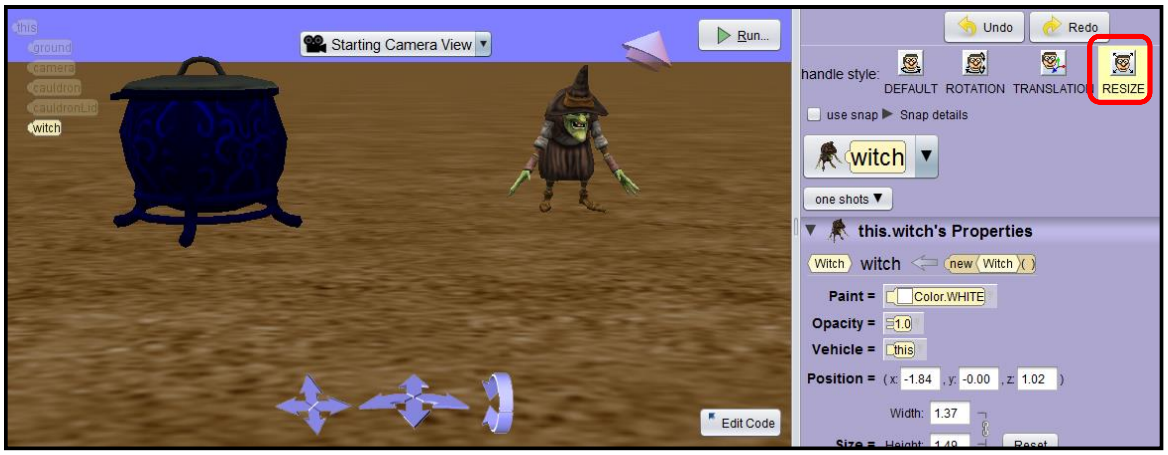
3. You should have a cauldron and a cauldron lid in your scene from exercise 1. Click **Scene Setup** so that we can practice some more with manipulating objects.



4. Let's add a **witch** to the scene. There are multiple ways to find her. You can search for her, click on hierarchy and biped class, click on theme and fantasy, or click on group and characters. Instead of clicking on the witch to add her to the scene, hold down your left mouse button on the **new Witch()** and drag onto the scene and release where you want her. You will notice a yellow bounding box on the screen indicating where she is going to appear. Leave the default name for her. If this method of adding objects does not work for you, then you can add the object by clicking on it and then moving it once it is added to the scene.



5. Make sure that the **witch** is selected on your scene and then click on **resize**. You should notice an arrow above the witch's head. This will resize the witch proportionally if you hold down your left mouse button and drag up or down. Make her bigger. Whatever you think looks good.



this.witch's Properties

Witch witch ⇐ new Witch()

Paint = Color.WHITE

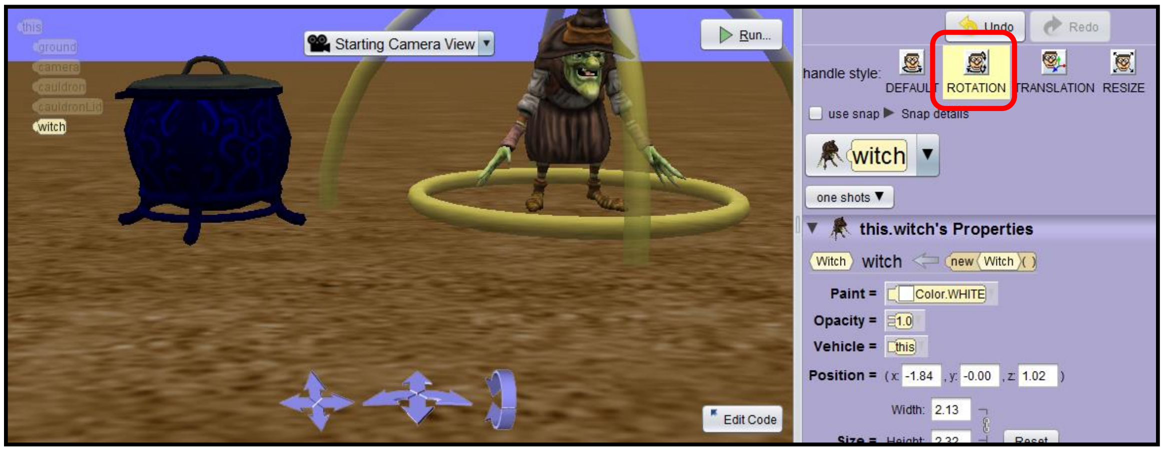
Opacity = 1.0

Vehicle = this

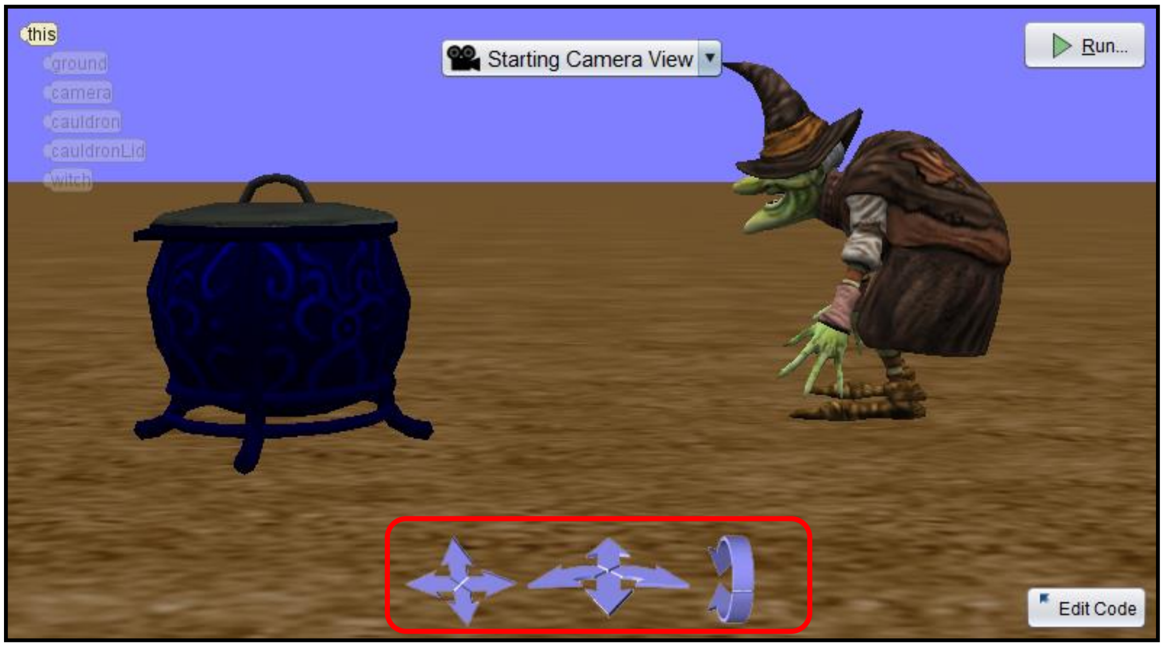
Position = (x: -1.84, y: -0.00, z: 1.02)

Width: 1.37

6. Now, let’s rotate her. Click on **rotation**. The up and down yellow lines will rotate the witch forwards and backwards if you hold down your left mouse button and drag. The yellow circle on the bottom of the witch will rotate the witch 360 degrees. Hold down your left mouse button on the bottom circle and drag to the left. Rotate the witch so that she is facing the cauldron.



7. Now, we are going to practice with the camera movements. The first set of arrows on the left will move the scene up, down, left, and right. The second set of arrows will move the witch forwards and backwards. The last set of arrows will adjust the scene up and down (more or less sky). Please try out each of the camera arrows.



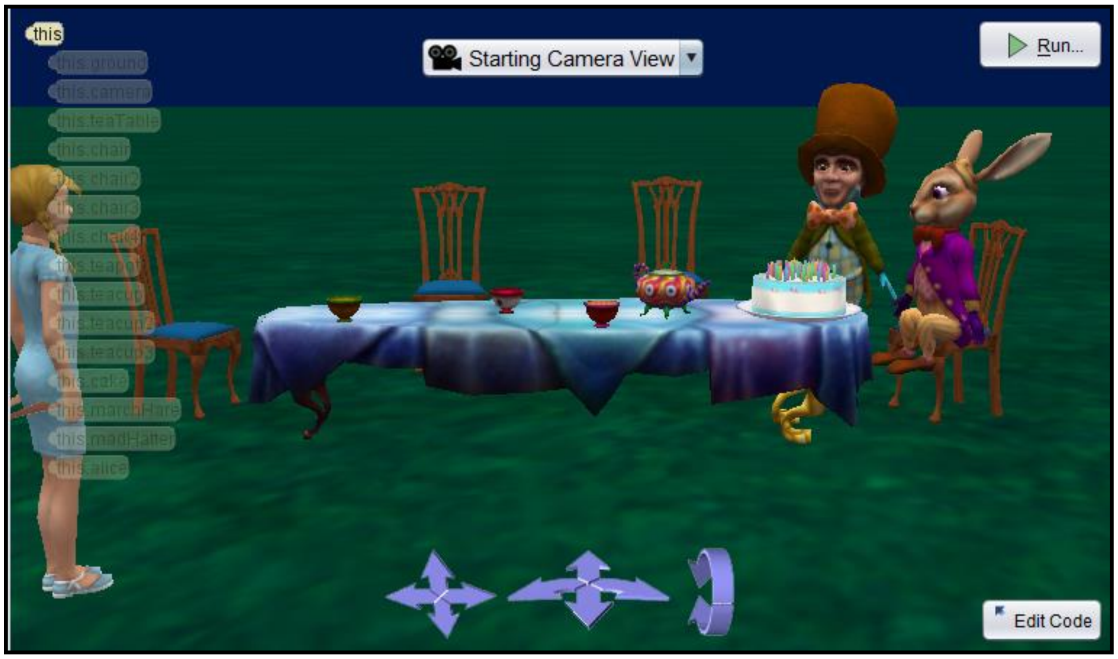
8. Please take some time to add more objects and manipulate those objects. Practicing with the environemnt is the best way to get acquainted with it. Save your work.

Alice in Wonderland Mad Tea Party Scene Setup

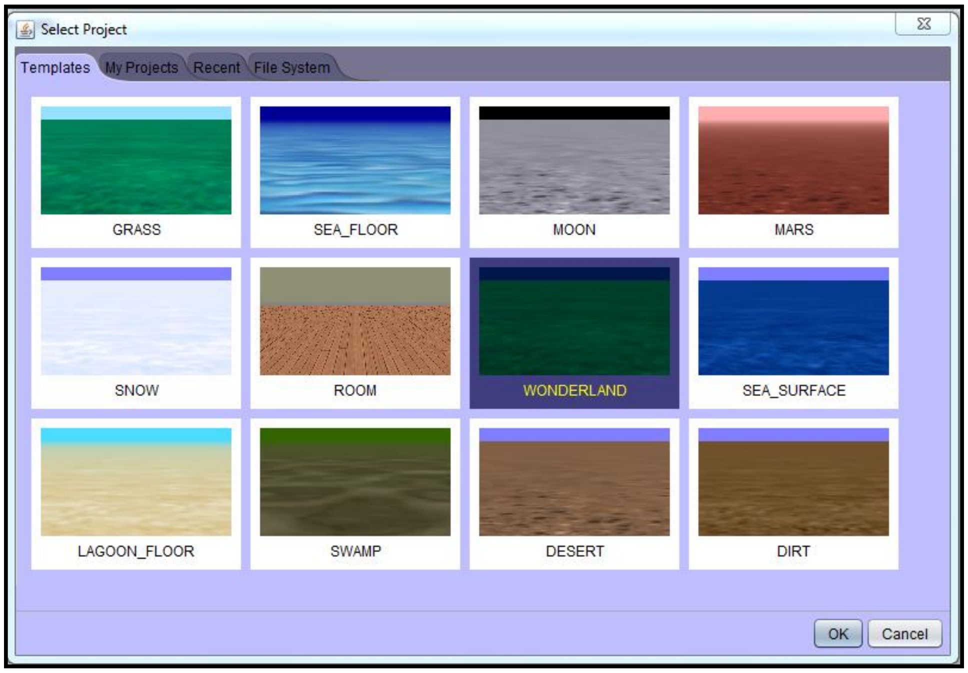
1. Open up Alice3. You will need to find the installed Alice3 folder and double click on the **Alice3.bat** batch file (alice3.sh for the Mac). A black window with command line will appear on your screen and should remain open (can be minimized) while Alice3 software is running.

2. The first step in programming is understanding the problem. We would like to create a trimmed version of the Alice in Wonderland unbirthday tea party. Once you understand the problem, you setup the scene and create a storyboard for animating the scene.

3. Our goal for the scene setup is to have the following characters: Alice, Mad Hatter, and the March Hare. We will also add some objects to make the scene more interesting: a table, chairs, a tea pot, tea cups, and a birthday cake. When we are finished it should look similar to the following:

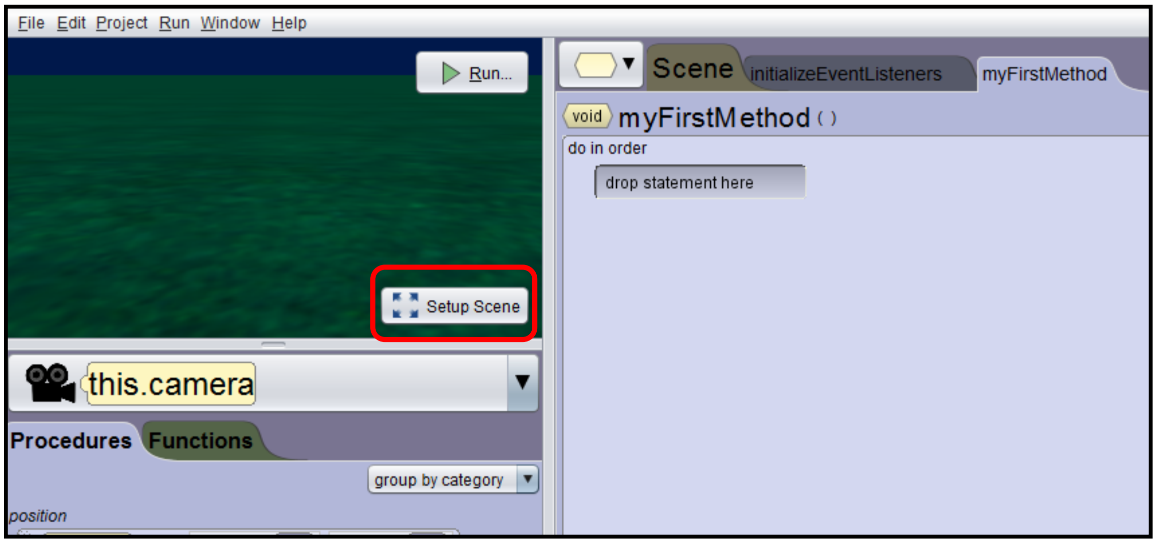


4. Select the **wonderland** template:

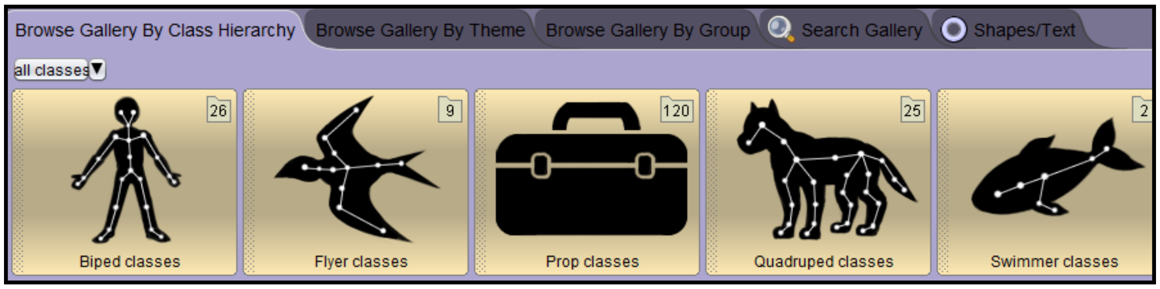


5. Select **File** from the menu, then **Save As**. Save this file as **TeaParty**. Please get in the habit of capitalizing the first letter of every word in your filename and do not use spaces when naming your files. You should save your work often. You can click Save from the File menu from this point on.

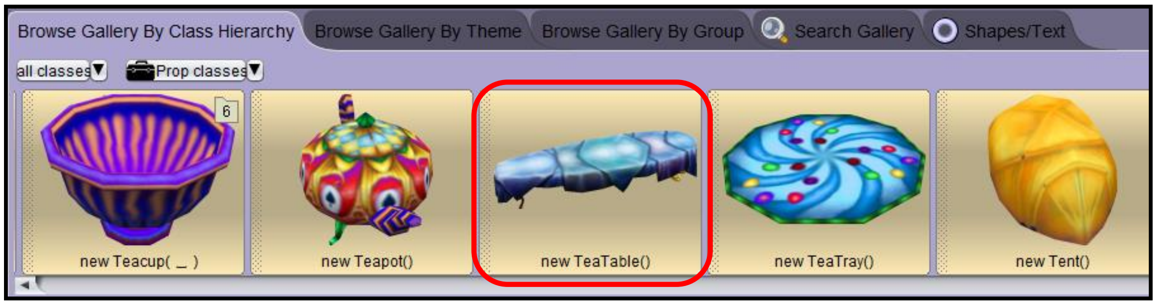
6. Click on **Setup Scene button.**



7. We are going to add a table to the scene for the characters to gather around. There is a tea table specifically designed for Alice in Wonderland. Click on the tab called **Browse Gallery By Class Hierarchy**.

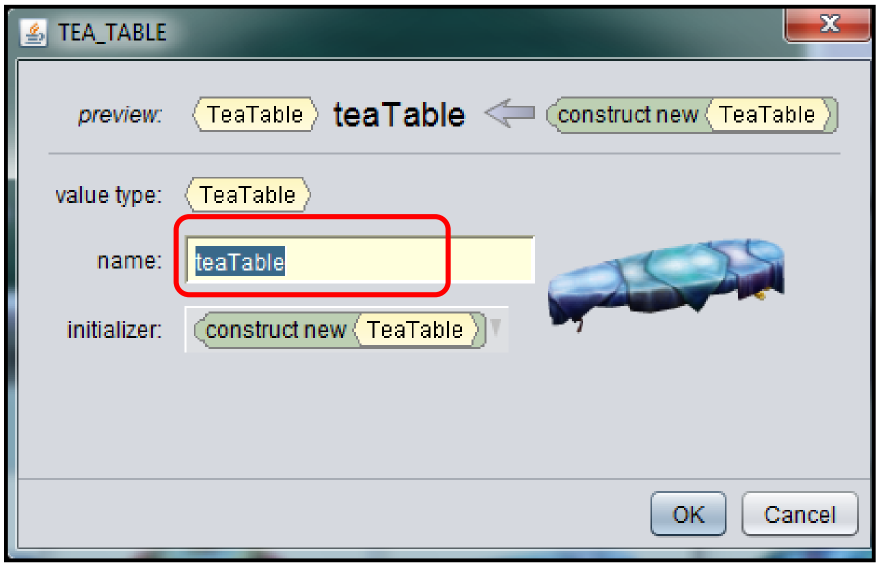


8. Click on the **Prop classes** category. Scroll to the end (they are in alphabetical order) until you see the **TeaTable** class. You could have used the ***Search Gallery*** tab to find the table as well.

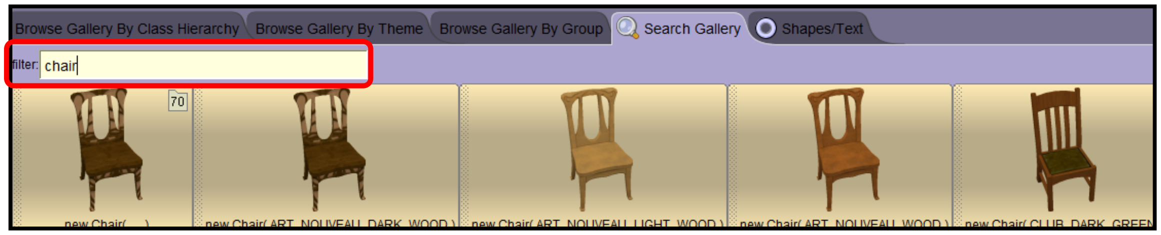


9. Click on the **TeaTable** class to add a tea table to your world or hold down your left mouse button and drag this object to wherever you would like to place it in your scene. If you choose to click on the TeaTable class, the new object will be placed automatically in the center of the scene.

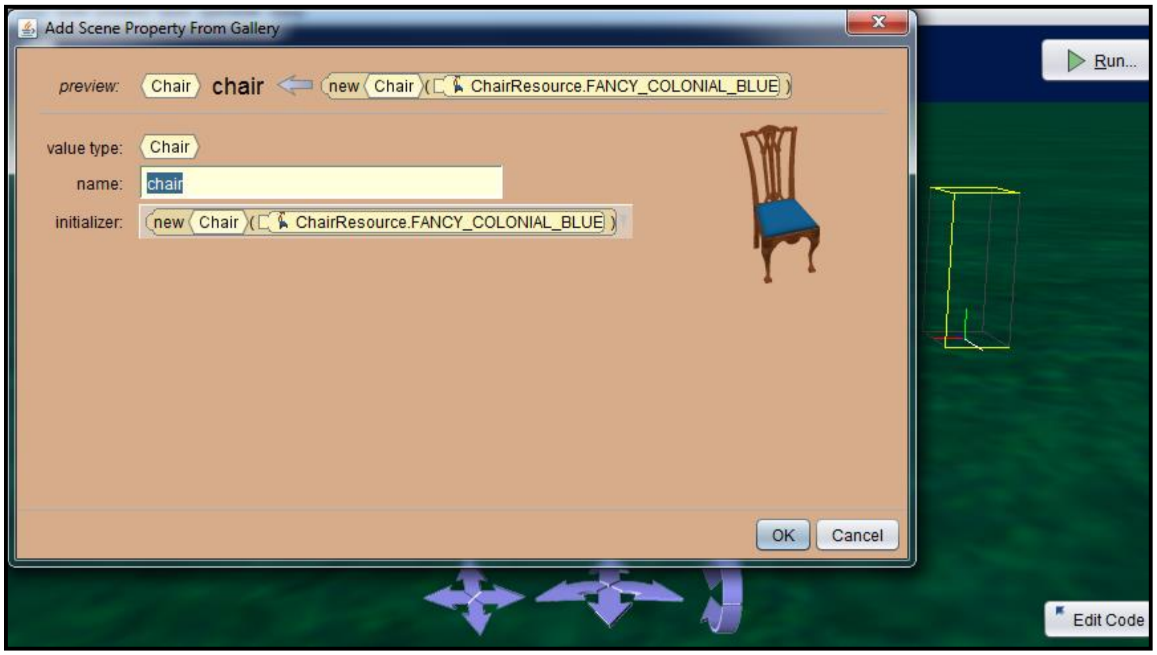
10. When you click on the class it will ask you for a name for the object. You can leave the name teaTable or rename if you want. Do not put spaces in your object name and the first letter of your object name should begin with a lowercase letter and the first letter of the second word should be a capital letter.


TEA_TABLE
preview: TeaTable teaTable construct new TeaTable
value type: TeaTable
name: teaTable
initializer: construct new TeaTable
OK
Cancel

11. Next, we are going to **add a chair**. Now we can test out the search feature in Alice by typing chair into the search box. You will have a list of all the chair models. Please choose the chair that you like.


Browse Gallery By Class Hierarchy
Browse Gallery By Theme
Browse Gallery By Group
Search Gallery
Shapes/Text
filter: chair
70
new Chair(...)
new Chair(ART_NOUVEAU_DARK_WOOD)
new Chair(ART_NOUVEAU_LIGHT_WOOD)
new Chair(ART_NOUVEAU_WOOD)
new Chair(CLUB_DARK_GREEN

12. Drag the chair that you want onto the scene where you want it by holding down the left mouse button and dragging from the class that you are choosing to add. You will see a yellow bounding box that shows you were your new object will be placed. When you get the object where you want it, release and it will ask you for a name for the object.



preview: Chair chair ⇐ new Chair(ChairResource.FANCY_COLONIAL_BLUE)

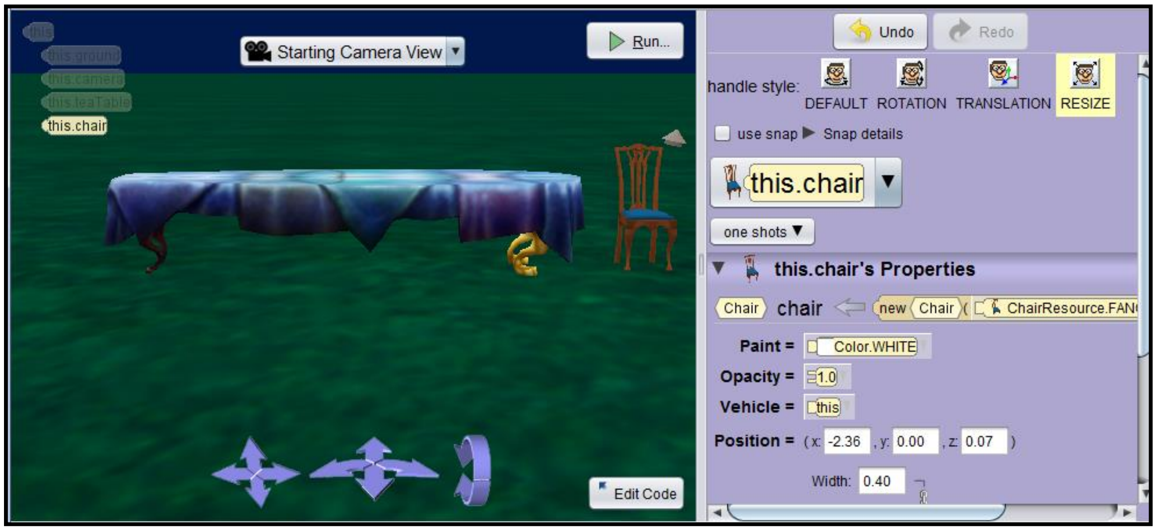
value type: Chair

name: chair

initializer: new Chair(ChairResource.FANCY_COLONIAL_BLUE)

You should name this object something simple. Let's call it **chair**.

13. We should resize the chair so that it matches the size of the table. To do this, you will need to select the chair and then click on the **resize** button from the handle style choices. When you click on the button, an arrow will appear above the chair. Holding down your left mouse button on the arrow and move your mouse up and down to resize the chair.



handle style: RESIZE

use snap ▶ Snap details

this.chair

this.chair's Properties

Chair chair ⇐ new Chair(ChairResource.FAN

Paint = Color.WHITE

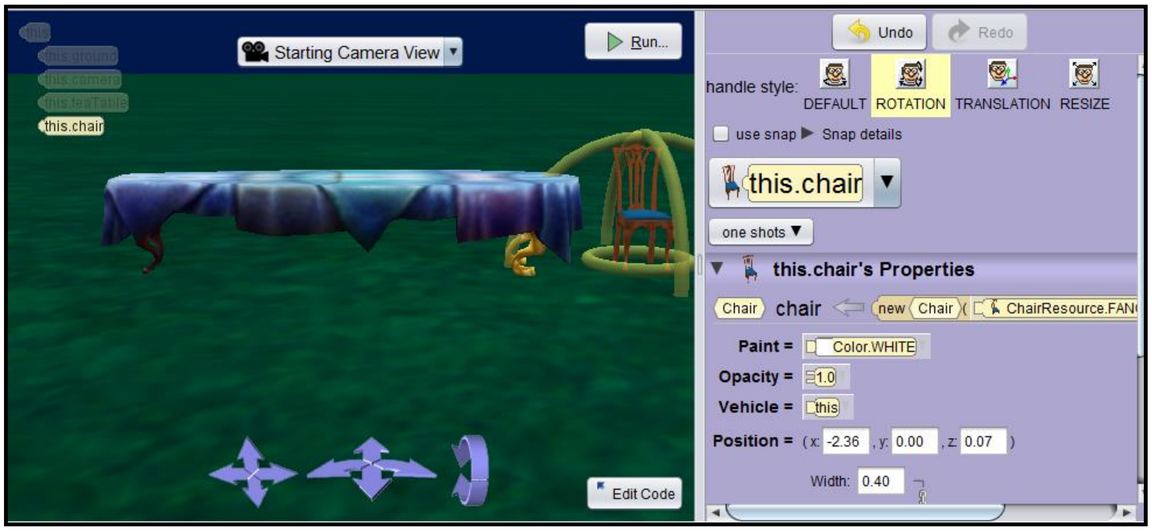
Opacity = 1.0

Vehicle = this

Position = (x: -2.36 , y: 0.00 , z: 0.07)

Width: 0.40

14. To rotate the chair, click on the **rotation** button from the handle style choices. If you hold down the left mouse button on the bottom ring and drag to the right and left, it will spin the chair around so that you can have it faces the table.



this.chair's Properties

Chair chair ⇐ new Chair(ChairResource.FAN

Paint = Color.WHITE

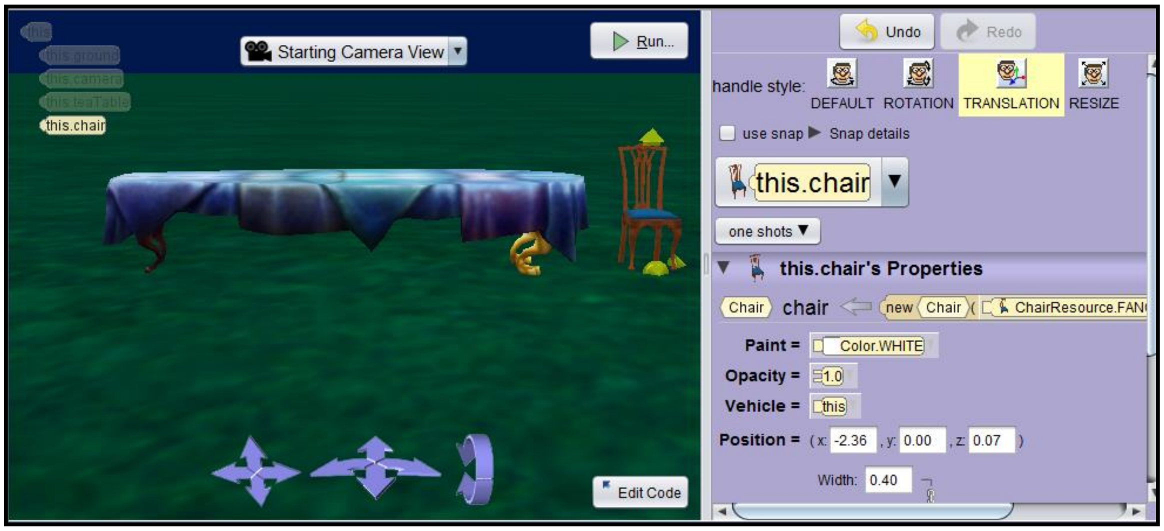
Opacity = 1.0

Vehicle = this

Position = (x: -2.36 , y: 0.00 , z: 0.07)

Width: 0.40

15. To move the chair, click on the **translation** button from the handle style choices. If you hold down your left mouse button on the arrow on top of the chair and drag up and down, the chair will move up and down. The arrow in the front will move the chair forward and backward. The arrow to the right will move the chair left and right.



this.chair's Properties

Chair chair ⇐ new Chair(ChairResource.FAN

Paint = Color.WHITE

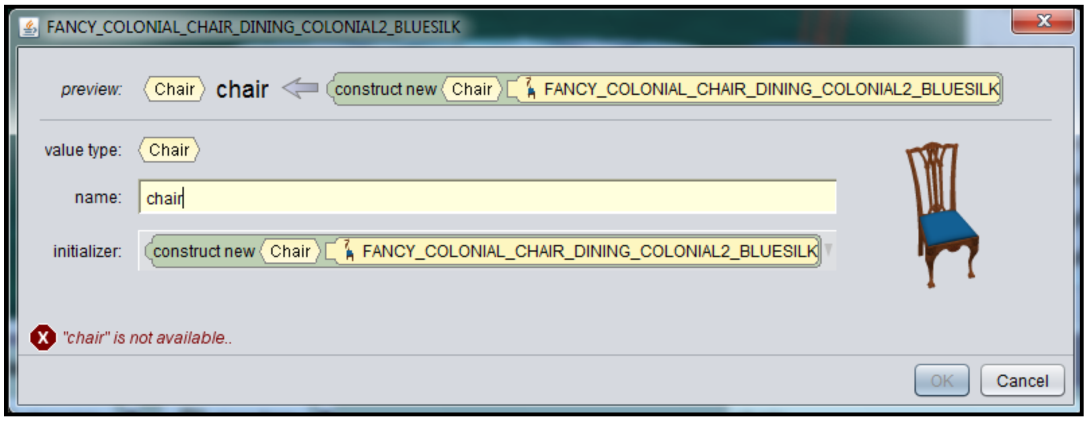
Opacity = 1.0

Vehicle = this

Position = (x: -2.36 , y: 0.00 , z: 0.07)

Width: 0.40

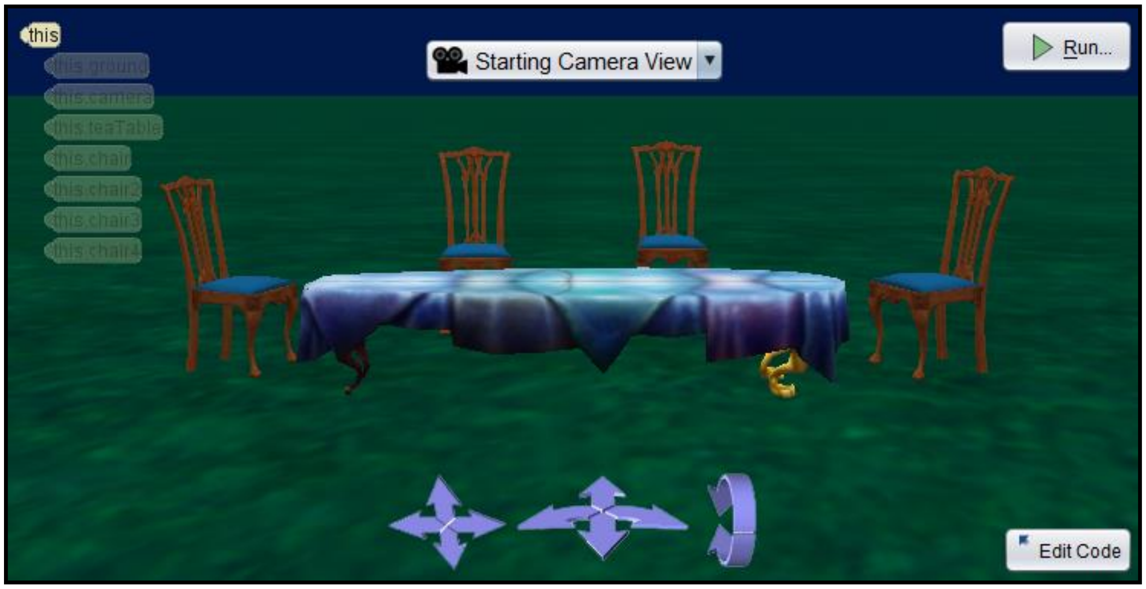
16. **Add 3 more chairs** to the scene around the table. Be careful not to give the chairs the same name. You will see the following error if you try to name your objects the same name. You can call the other chairs: chair2, chair3, and chair4. Do not put spaces in your names. The Alice software will not allow you to name your objects with spaces and this is because the Java language does not allow you to have spaces when naming.


FANCY_COLONIAL_CHAIR_DINING_COLONIAL2_BLUESILK
preview: Chair chair construct new Chair FANCY_COLONIAL_CHAIR_DINING_COLONIAL2_BLUESILK
value type: Chair
initializer: construct new Chair FANCY_COLONIAL_CHAIR_DINING_COLONIAL2_BLUESILK
OK
Cancel

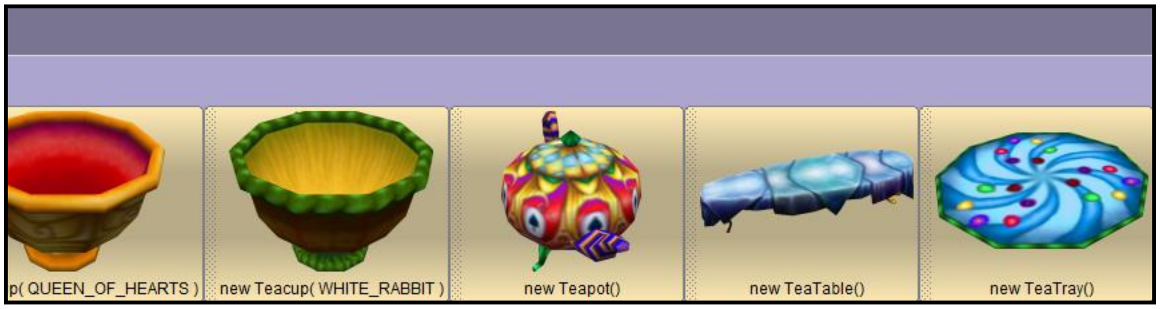
name: chair

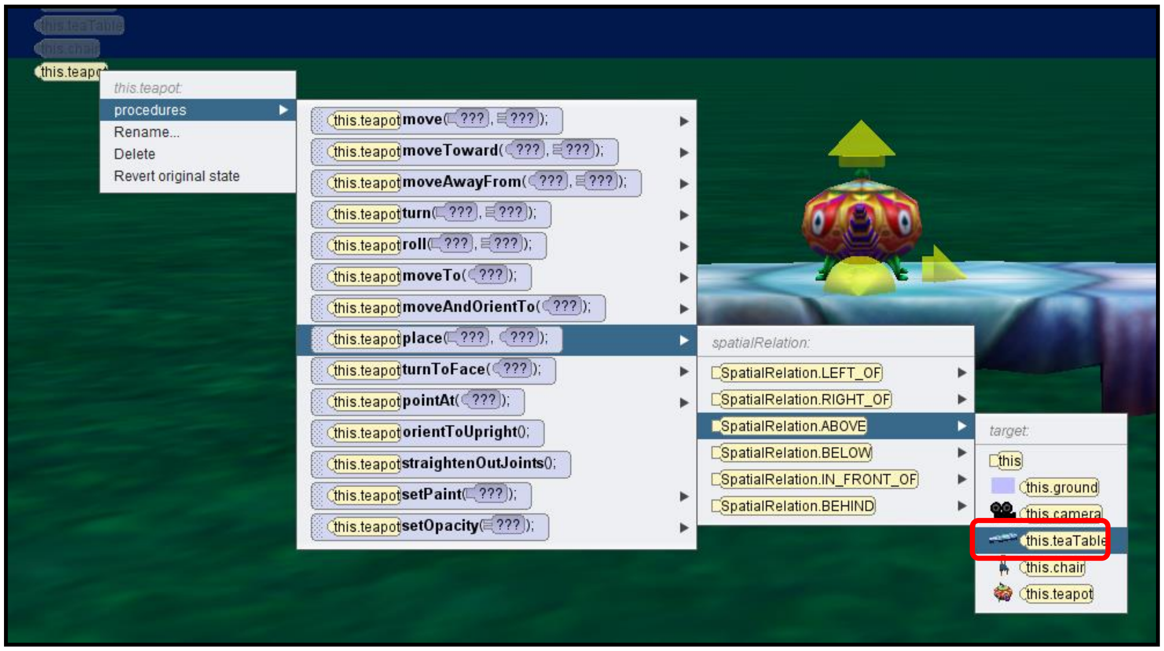
"chair" is not available..

17. It should look similar to the following:



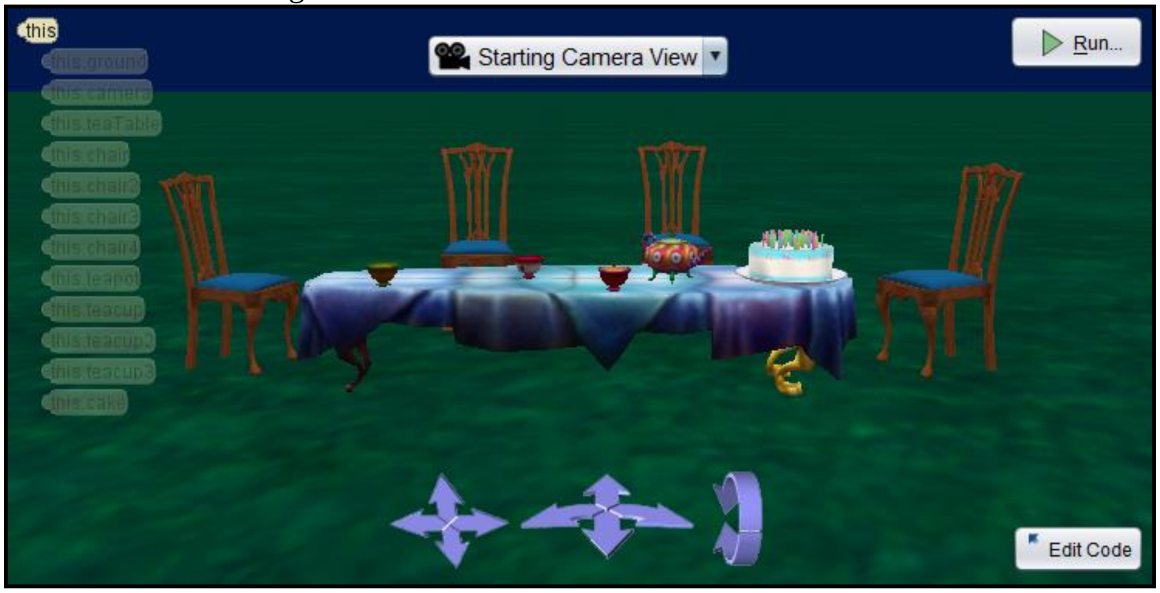
18. Next, we need to **add some teacups and a teapot onto the table**. If you search for tea in the gallery, you will be given the teapot, teacups, saucers, etc. I would like to start with the teapot. When you create the teapot, you can use the default name. We can play with trying to get this teapot onto the table, but this would take a while and there is an easier way. If you **right click on the teapot**, select **procedures**, **teapot place…**, **above**, and **teaTable**, it will place the teapot on top of the table for you.



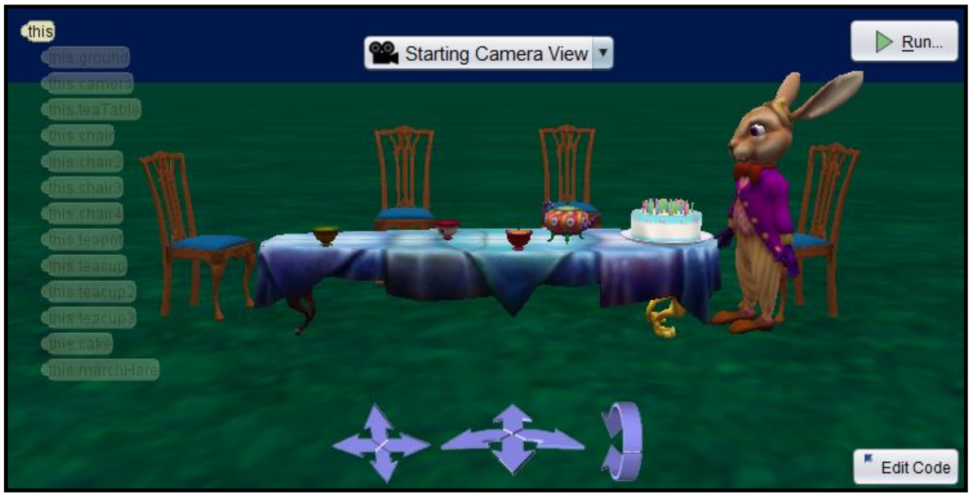

this.teaTable
this.chair
this.teapot
this.teapot:
procedures
Rename...
Delete
Revert original state
this.teapot move(???, ???);
this.teapot moveToward(???, ???);
this.teapot moveAwayFrom(???, ???);
this.teapot turn(???, ???);
this.teapot roll(???, ???);
this.teapot moveTo(???);
this.teapot moveAndOrientTo(???);
this.teapot place(???, ???);
this.teapot turnToFace(???);
this.teapot pointAt(???);
this.teapot orientToUpright();
this.teapot straightenOutJoints();
this.teapot setPaint(???);
this.teapot setOpacity(???);
spatialRelation:
SpatialRelation.LEFT_OF
SpatialRelation.RIGHT_OF
SpatialRelation.ABOVE
SpatialRelation.BELOW
SpatialRelation.IN_FRONT_OF
SpatialRelation.BEHIND
target:
this
this.ground
this.camera
this.teaTable
this.chair
this.teapot

19. **Add a few teacups onto the table** and adjust them how you want them. *Be careful not to give 2 teacups the same name.*

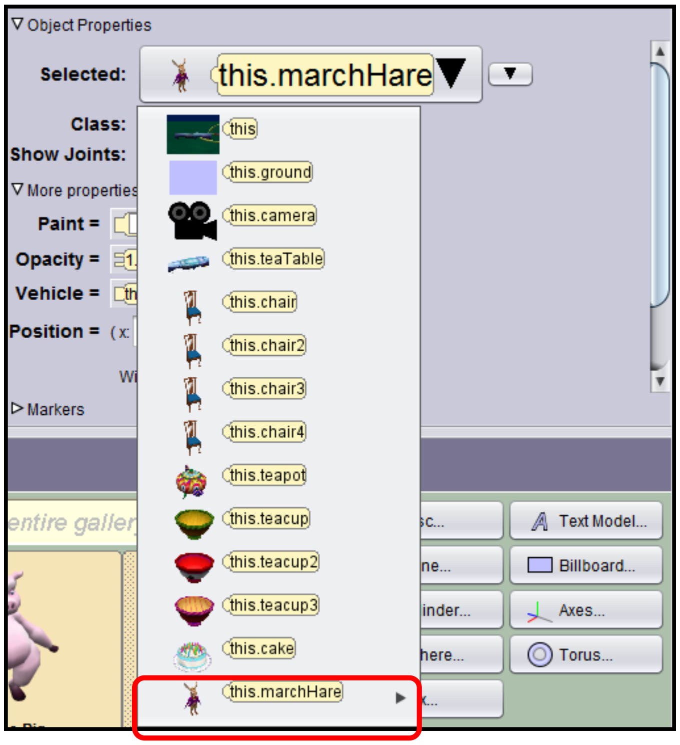
20. **Add a birthday cake onto the table** and readjust the items on the table. It should look similar to the following.

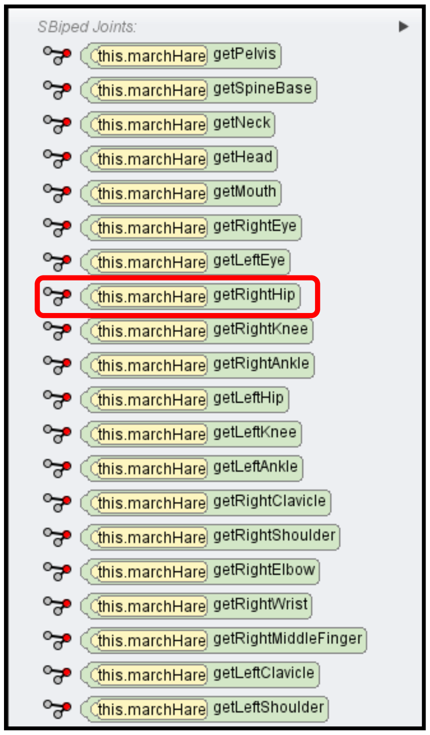


21. Next, we are going to add the characters. The characters can be found in the biped folder in the gallery. Let’s add the March Hare first. Place him directly in front of one of the chairs. It doesn’t matter which chair you choose. You will need to rotate him so that he lined up with the chair. We are going to make him sit in the chair.

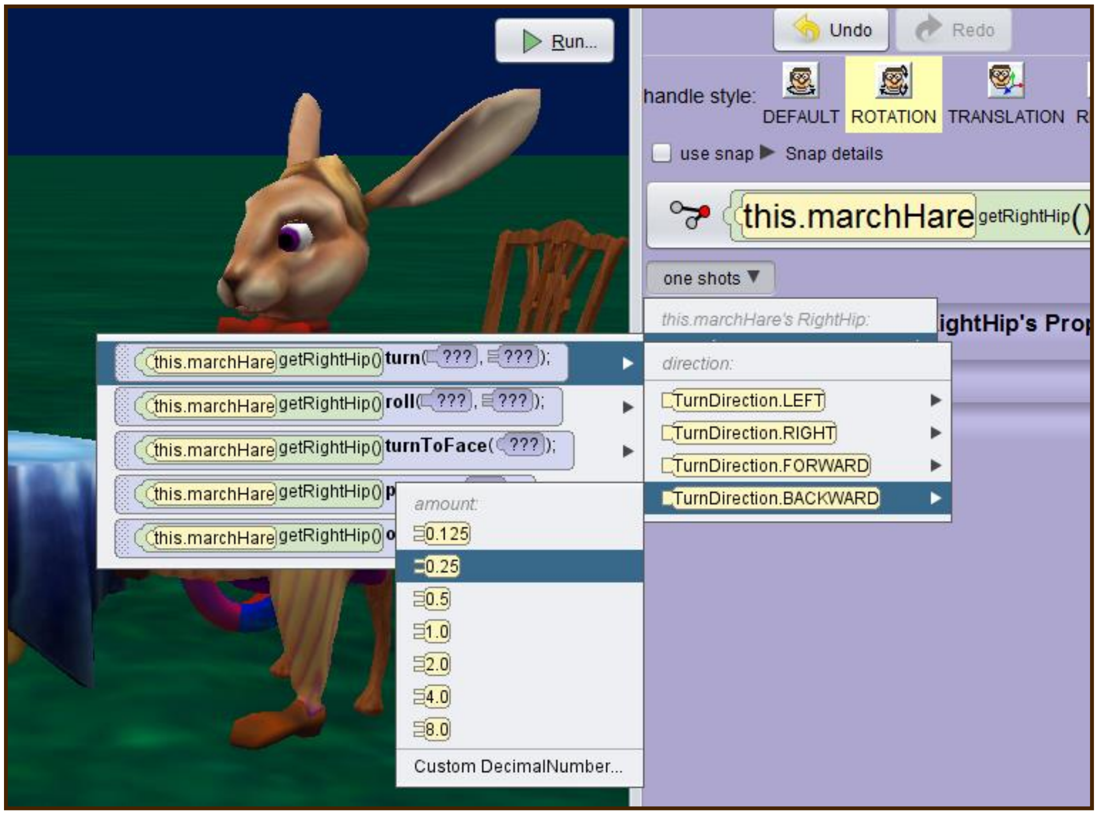


22. To move the marchHare's joints, we will need to select the marchHare and drop down his subparts as shown below. Choose the hare's right hip.




SBiped Joints:
this.marchHare getPelvis
this.marchHare getSpineBase
this.marchHare getNeck
this.marchHare getHead
this.marchHare getMouth
this.marchHare getRightEye
this.marchHare getLeftEye
this.marchHare getRightHip
this.marchHare getRightKnee
this.marchHare getRightAnkle
this.marchHare getLeftHip
this.marchHare getLeftKnee
this.marchHare getLeftAnkle
this.marchHare getRightClavicle
this.marchHare getRightShoulder
this.marchHare getRightElbow
this.marchHare getRightWrist
this.marchHare getRightMiddleFinger
this.marchHare getLeftClavicle
this.marchHare getLeftShoulder

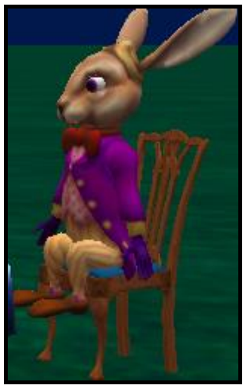
Now, we need to select **ONE SHOT, procedures**, **marchHare.getRightHip.turn…**, **BACKWARD**, and **0.25**



23. **Repeat this for the leftHip**.

24. Select the **marchHare's rightKnee**, then select **one shots**, **procedures**, **turn**, **forward**, and **0.25**.

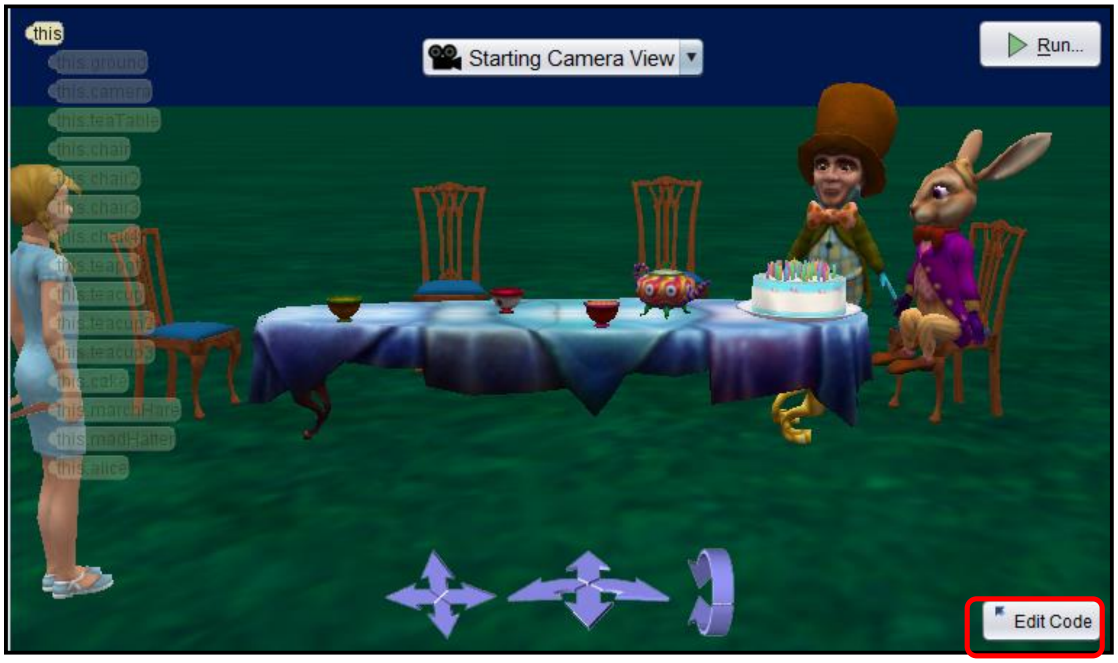
25. **Repeat this for the leftKnee**. You may need to move the entire marchHare back and up to get him onto the chair.



26. Now, let's **add the madHatter** to the scene. Place him next to the marchHare. It doesn't matter which side he is on. You may need to resize, rotate, and move him to get the scene to look the way you want.

27. Finally, we are going to add Alice to the scene. We will need to create Alice using the Child class in the biped classes. The Child class allows you to select male or female, the skin tone, the attire, the hair color, eye color, and shape of the person. **Create a girl** that looks like Alice and name her **alice**. Normally you would capitalize a name, but when we name objects, we don't capitalize the object names.

28. Place alice off to the side of the animation window looking at the tea party as shown below.



29. We are finished with the scene setup. If you want to add some wonderland trees or other objects to your scene, feel free.

30. Save this program and exit Alice.

Making an Alien Walk in Alice

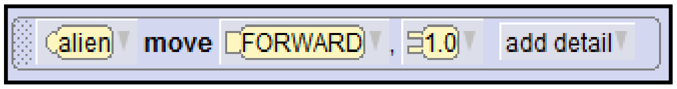
We are going to using a variable to make an alien walk in Alice.

1. Get into Alice. Choose the **mars** template.

2. Choose Setup Scene**. Add** an **alien** object from the Biped folder. You may want to rotate the alien to the side so that you can see the leg movements easier.

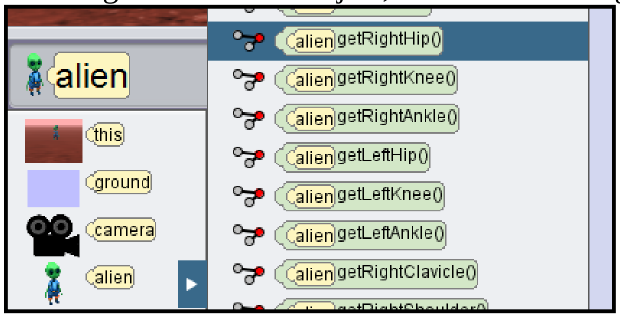
3. Save this program as **AlienWalk**.

4. Click on **Edit Code**. Select the alien and drag the **move** method to the editor. Choose forward and 1 meter *(these choices are known as arguments)* to fullfill the direction and amount questions. Click the **run** button to test your program.

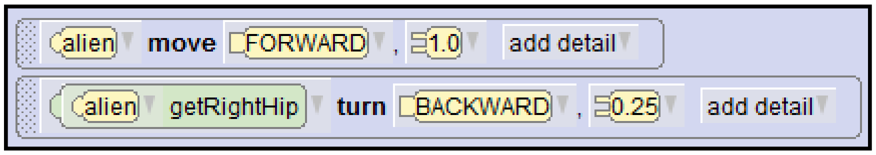


```
alien move FORWARD , 1.0 add detail
```

5. Now select the alien's right hip, by clicking on the object drop down, selecting the arrow to the right of the alien object, and then choosing **getRightHip()**

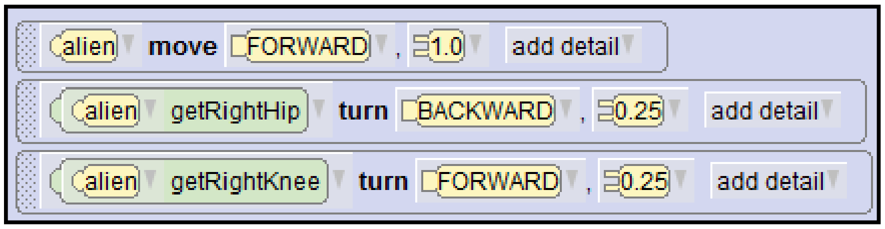


6. Next, drag the **turn** method for the right hip onto the editor under the move method. Choose **backward** and **0.25** as your arguments. Think of the directions for turning a subpart of an object as clockwise for forward motion and counterclockwise for backward motion. The 0.25 represents the amount that the subpart will turn *(1 would be 1 full revolution, therefore 0.25 would ¼th of the way around)*. Click the **run** button to test your program.



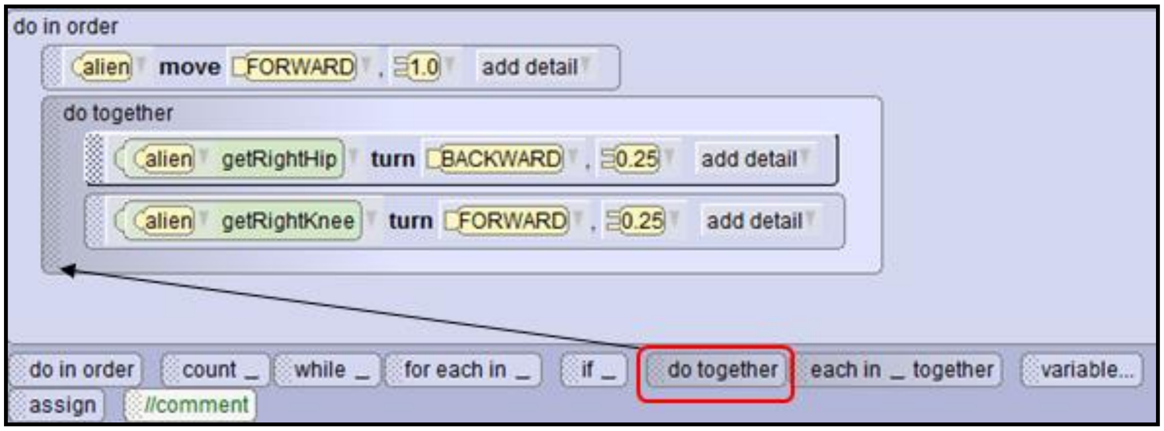
```
alien move FORWARD , 1.0 add detail
(alien getRightHip) turn BACKWARD , 0.25 add detail
```

7. Now let’s have the right knee turn forward 0.25. Remember to think closewise for forward motions and counterclockwise for backwards motions.



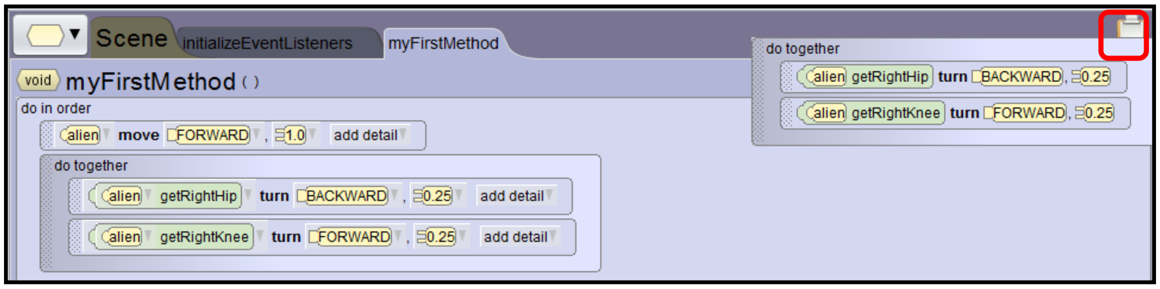
```
alien move FORWARD , 1.0 add detail
alien getRightHip turn BACKWARD , 0.25 add detail
alien getRightKnee turn FORWARD , 0.25 add detail
```

8. Click the **run** button to test your program. This walk looks akward. We need to fix the timing of these movements. Drag the **do together** block underneath the move method and then drag the right hip and right knee turn methods in the do together block so that it happen simultaneously. Run your animation again. This should look smoother.



```
do in order
    alien move FORWARD , 1.0 add detail
    do together
        alien getRightHip turn BACKWARD , 0.25 add detail
        alien getRightKnee turn FORWARD , 0.25 add detail
```

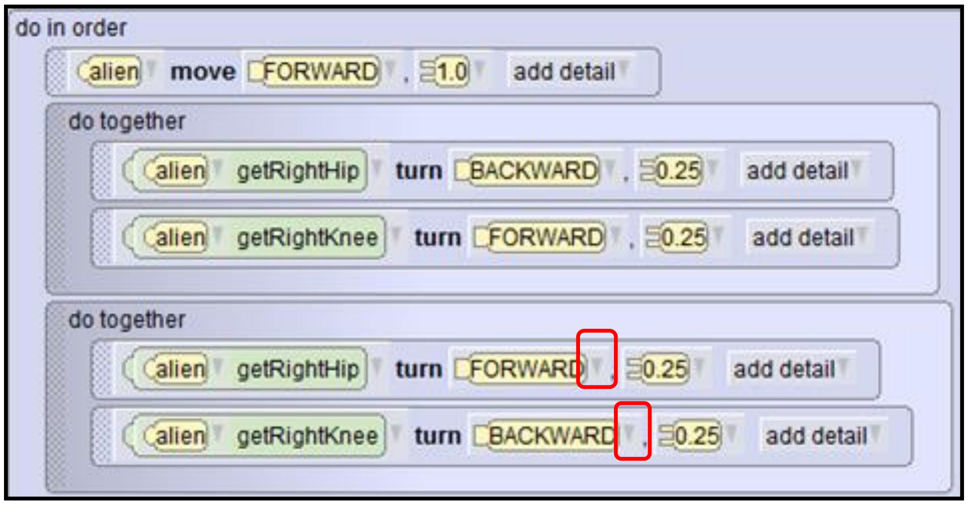
9. Since this leg is raised up, we need to put it back down to create a walking motion. We are going to practice copying the do together block that we already have written to the clipboard. Hold down the **ctrl button** *(command button on the Mac)* and **drag** the **do together block** to the **clipboard** in the right hand corner and release. You should now see a white piece of paper on the clipboard *(this indicates that you copied code to the clipboard)*. If you hover your mouse over the clipboard, you can see the code. *(Note: if you do not hold the ctrl button down when you added code to the clipboard, it will cut instead of copying. Click Edit > Undo from the menu if you did a cut.)*



```
void myFirstMethod ( )
do in order
    alien move FORWARD , 1.0 add detail
    do together
        alien getRightHip turn BACKWARD , 0.25 add detail
        alien getRightKnee turn FORWARD , 0.25 add detail
```

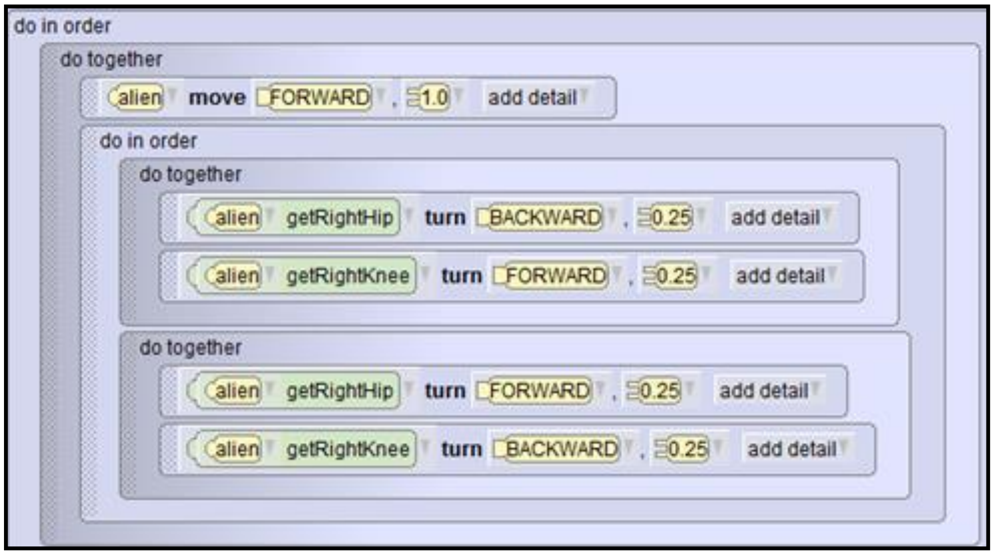
```
do together
    alien getRightHip turn BACKWARD, 0.25
    alien getRightKnee turn FORWARD, 0.25
```

10. Hold down the **Control key** (Command key on Mac) and **drag** the piece of paper from the clipboard to underneath your first do together block to create a second do together block. Make sure that the second do together block is not inside of the other block. Change the second do together block **right hip turn to forward** and the **right knee turn to backward** *(click the drop down next to the direction)* as shown below:



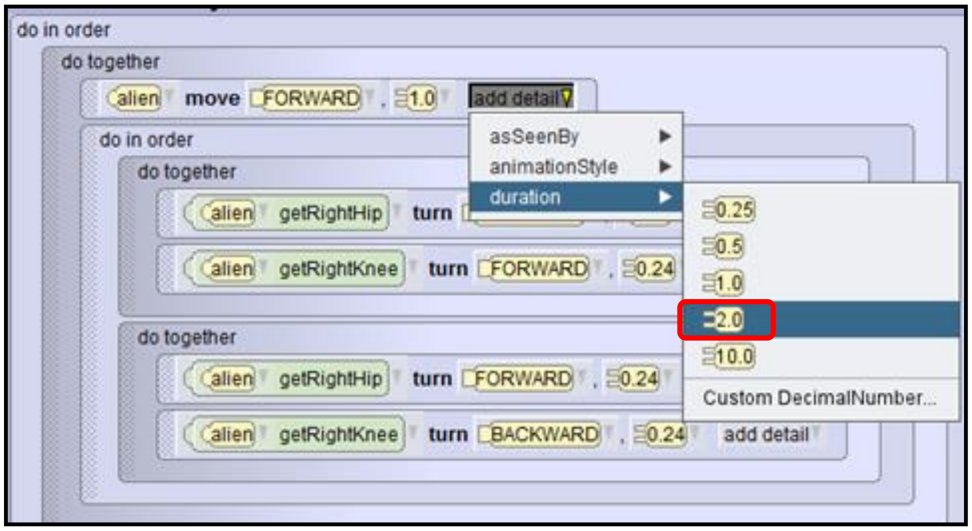
```
do in order
    alien move FORWARD , 1.0 add detail
    do together
        alien getRightHip turn BACKWARD , 0.25 add detail
        alien getRightKnee turn FORWARD , 0.25 add detail
    do together
        alien getRightHip turn FORWARD , 0.25 add detail
        alien getRightKnee turn BACKWARD , 0.25 add detail
```

11. It looks silly that the alien moves forward before his right leg bends. We need him to move forward at the same time that he moves his legs. This is a bit challenging because if we add the move to the first do together, then he will be done moving forward before he puts his leg back down. If we combine the 2 do togethers, then he won't bend his leg at all because the forward and backwards motions will cancel each other out. We need to **move the 2 do together blocks** into a **do in order block** and then have the do in order block happening at the same time as the move method. This may take some practice. If you make a mistake click **Edit** on the menu bar and then **Undo**.



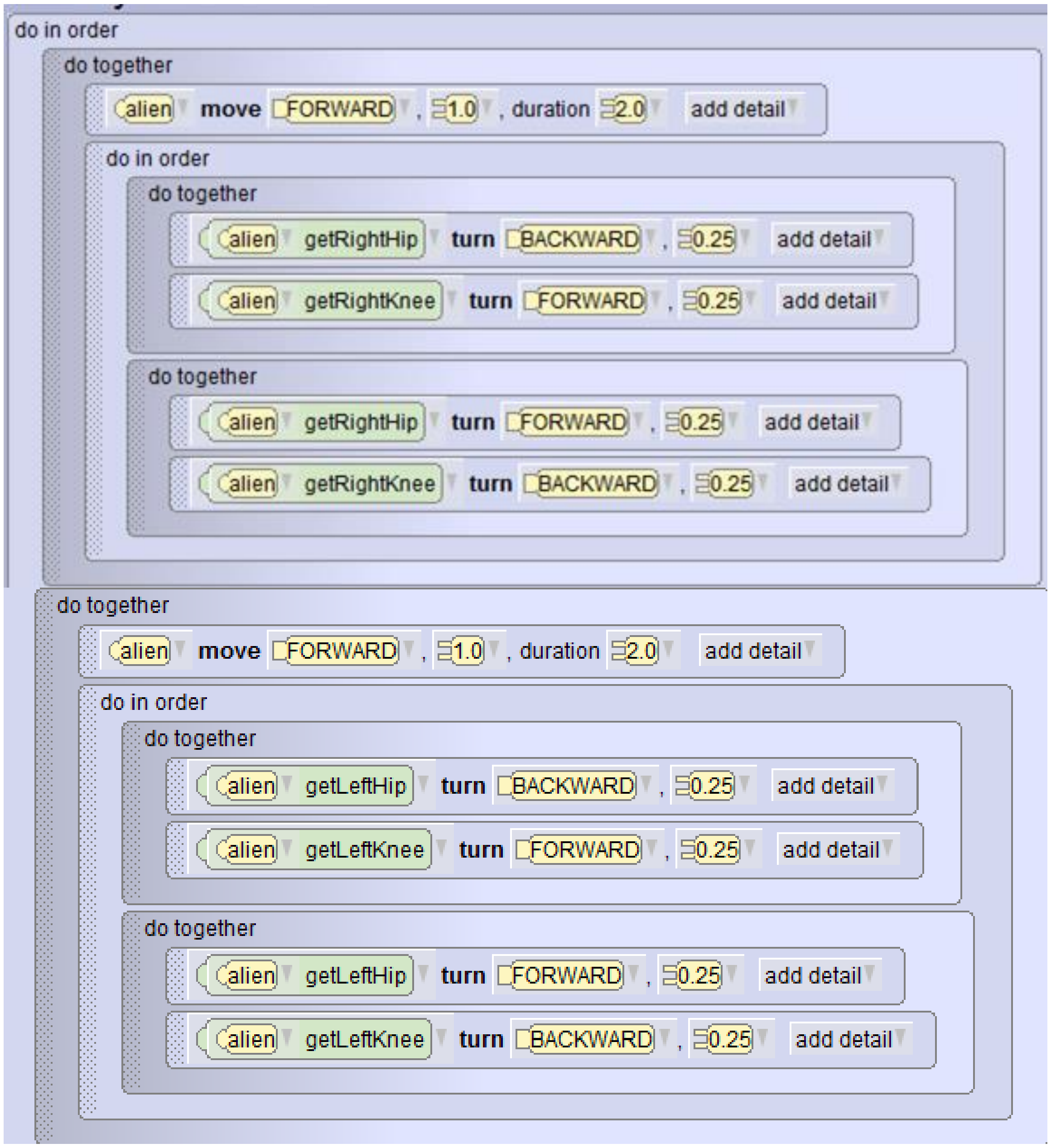
```
do in order
  do together
    alien move FORWARD , 1.0 add detail
    do in order
      do together
        alien getRightHip turn BACKWARD , 0.25 add detail
        alien getRightKnee turn FORWARD , 0.25 add detail
      do together
        alien getRightHip turn FORWARD , 0.25 add detail
        alien getRightKnee turn BACKWARD , 0.25 add detail
```

12. You may want to change the duration of the move to be 2 seconds instead of 1 second *(default)*. Since the first do together block will take 1 second and the second do together block will take 1 second, it would be nice if the move took the same amount of time as both do together blocks added together *(which would be 2 seconds).*



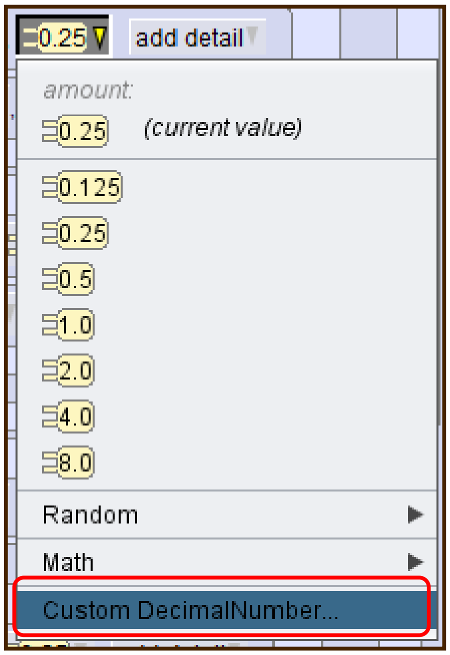
```
do in order
  do together
    alien move FORWARD , 1.0 add detail
    do in order
      do together
        alien getRightHip turn
        alien getRightKnee turn FORWARD , 0.24
      do together
        alien getRightHip turn FORWARD , 0.24
        alien getRightKnee turn BACKWARD , 0.24 add detail
```

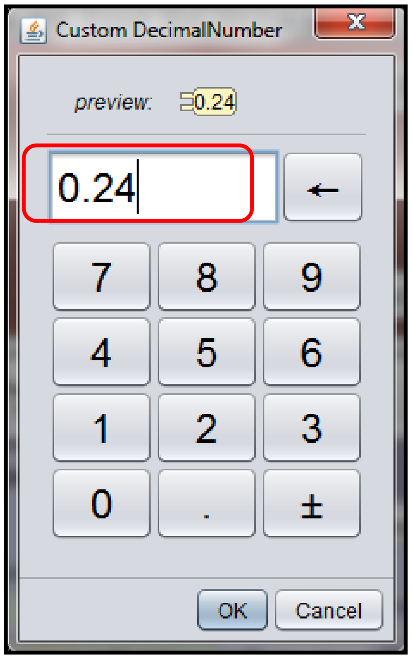
13. Repeat the code for the left hip and left knee to create a walking motion. *Hint: use the clipboard to copy the code that you already have and then change the body part to be left hip and left knee.* The code is shown below if you need some help.



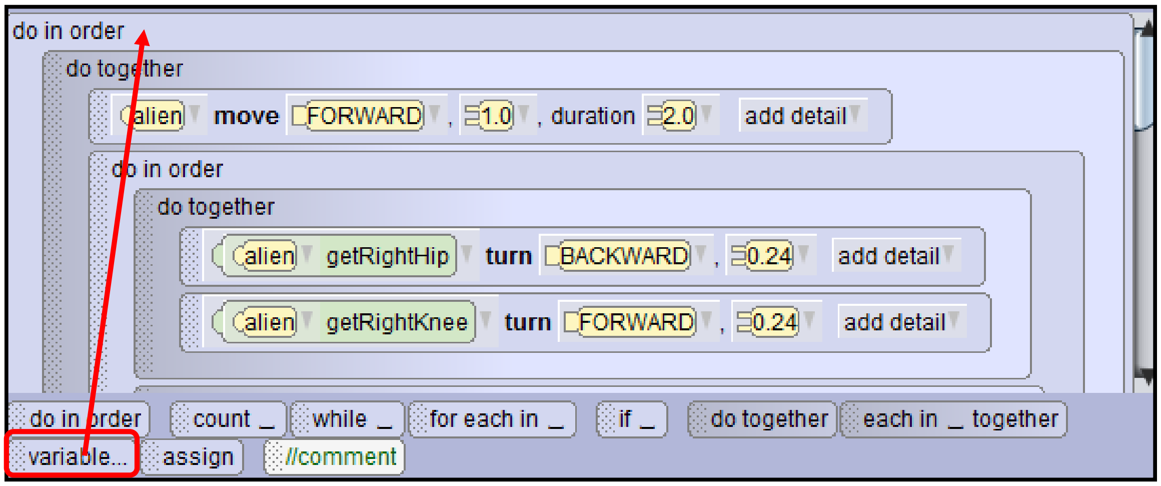
```
do in order
  do together
    alien move FORWARD , 1.0 , duration 2.0 add detail
    do in order
      do together
        alien getRightHip turn BACKWARD , 0.25 add detail
        alien getRightKnee turn FORWARD , 0.25 add detail
      do together
        alien getRightHip turn FORWARD , 0.25 add detail
        alien getRightKnee turn BACKWARD , 0.25 add detail
  do together
    alien move FORWARD , 1.0 , duration 2.0 add detail
    do in order
      do together
        alien getLeftHip turn BACKWARD , 0.25 add detail
        alien getLeftKnee turn FORWARD , 0.25 add detail
      do together
        alien getLeftHip turn FORWARD , 0.25 add detail
        alien getLeftKnee turn BACKWARD , 0.25 add detail
```

14. Run your animation. It looks good for now, but what if we decided that we wanted to adjust the amount that the alien bends his hip and knees? Let's try changing the amount to be 0.24 instead of 0.25. Select **Custom DecimalNumber…** and type **0.24**. You will need to do this for all 8 movements.





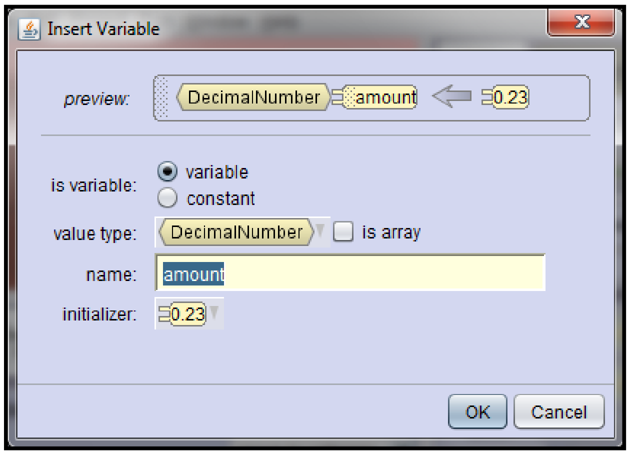
15. Hmmm….now let's say we want 0.23. It is such a pain to keep changing all 8 movements to test what looks good. It would be nice if there was an easier way to change this. Well you are in luck. If we create a variable to replace the 0.24, then we can just change the value of the variable instead of changing all the values every time. Drag the **variable block** to the top of your code editor.



```
do in order
    do together
        (alien) move FORWARD , 1.0 , duration 2.0  add detail
        do in order
            do together
                (alien getRightHip) turn BACKWARD , 0.24  add detail
                (alien getRightKnee) turn FORWARD , 0.24  add detail
```

16. Let's choose **DecimalNumber** as the type, **amount** as the variable name, and **0.23** as the intializer *(value)*. Click **OK**. The DecimalNumber is an Alice variable type, if you switch

your preferences to Java mode it will show as a Double variable type instead. To change your preferences, you would need to select **Window** from the menu, **Preferences**, **Programming Language**, **Java** *(Note: using Java view in Alice will make Do Together blocks of code look more challenging).*


Insert Variable

preview: DecimalNumber amount ⇐ 0.23

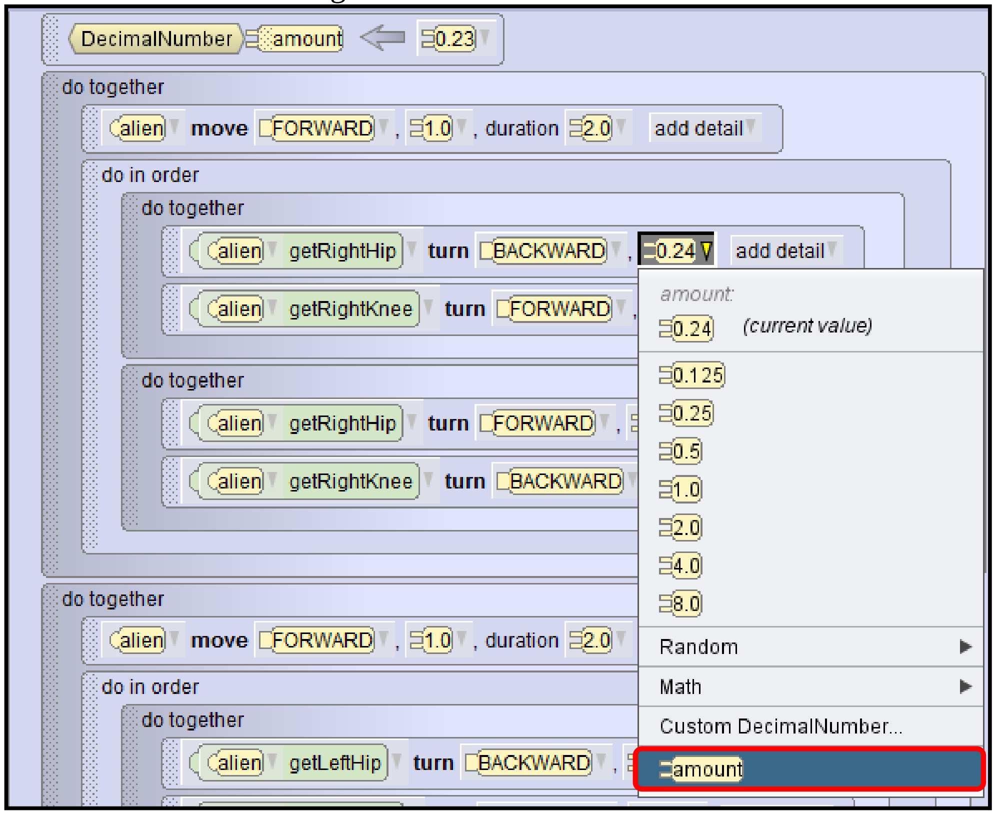
is variable: [✓ variable] constant

value type: DecimalNumber ☐ is array

name: amount

initializer: 0.23

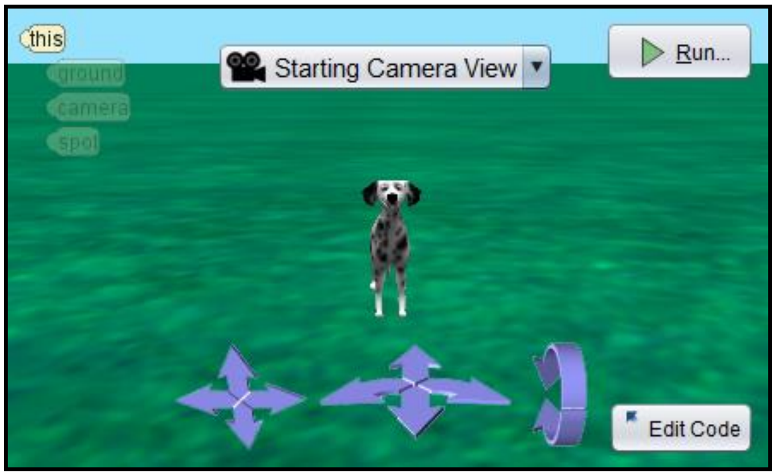
17. Now we will need to change the 0.24 to the **amount variable** for all 8 turn movements.


```
DecimalNumber amount ⇐ 0.23
do together
    alien move FORWARD , 1.0 , duration 2.0  add detail
    do in order
        do together
            alien getRightHip turn BACKWARD , 0.24  add detail
            alien getRightKnee turn FORWARD ,
        do together
            alien getRightHip turn FORWARD ,
            alien getRightKnee turn BACKWARD
do together
    alien move FORWARD , 1.0 , duration 2.0
    do in order
        do together
            alien getLeftHip turn BACKWARD ,
```

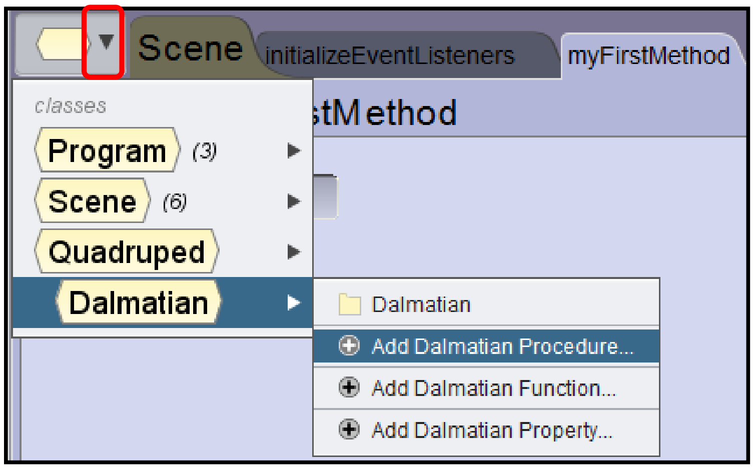
18. From this point on, we can change the value in the amount variable and it will update the value everywhere we used the amount variable. Save and close this program.

Making a Dog's Tail Wag in Alice

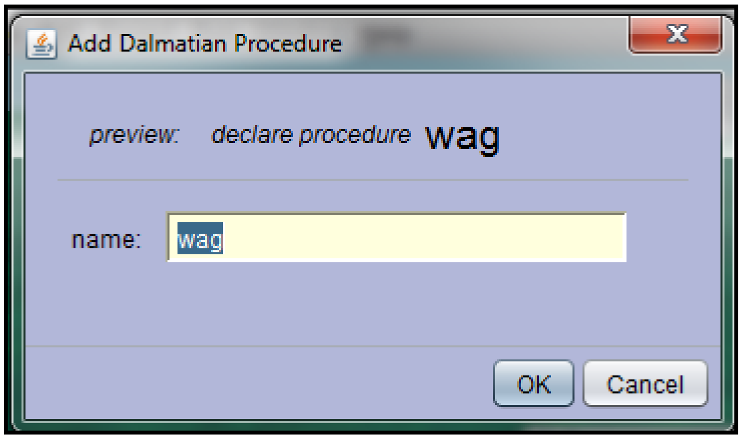
1. Get into Alice3.
2. Choose the **Grass** template.
3. Save the file as **DogWag**.
4. Click on **Setup Scene** button.
5. Add a Dalmatian to the scene. Name the dog **spot**. Then click on the **Edit Code** button.



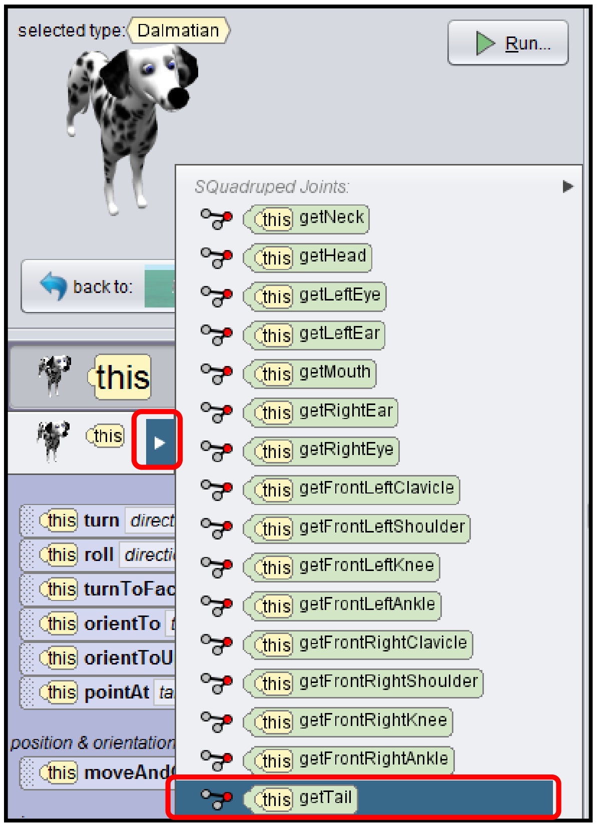
6. We would like to make the dog's tail wag, but instead of writing the code in myFirstMethod, we should add the code to the class that the dog belongs to. Since our dog object was created from the Dalmatian class, we would need to add our wag procedural method to the Dalmatian class. Click on the drop down arrow as shown below. Select the **Dalmatian** class and then select **Add Dalmatian Procedure…**



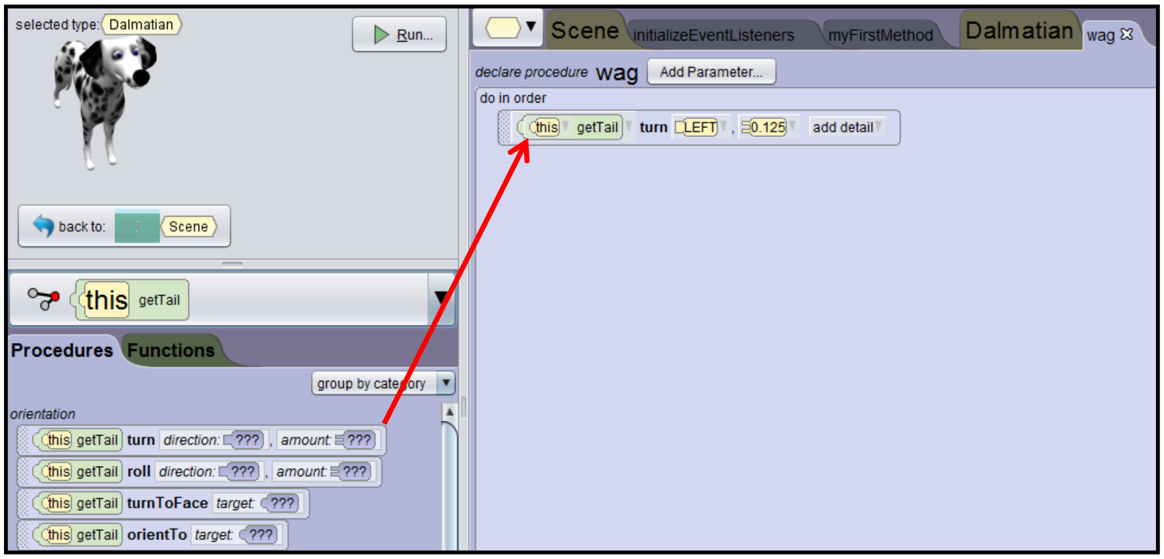
7. Name this new procedure **wag**.



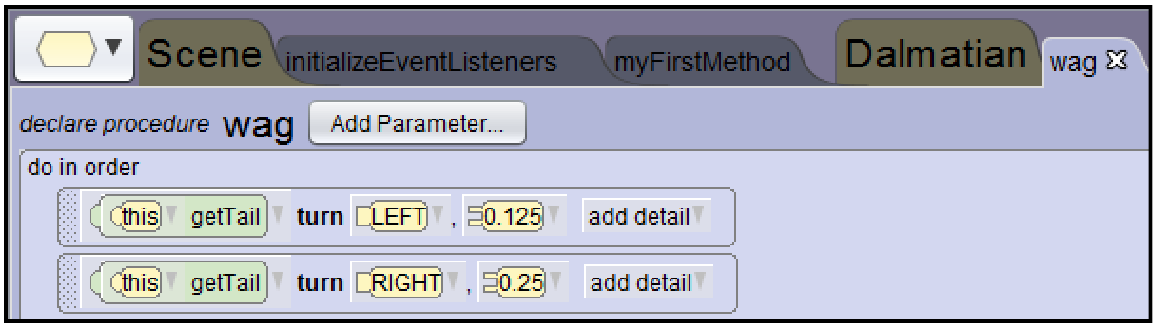
8. Select the subpart tail for the Dalmatian object. Click the drop down next to **this** in your object tree. Then click on the arrow to the right of the word **this** and select **this.getTail**. The word "this" represents the current class which is Dalmatian.



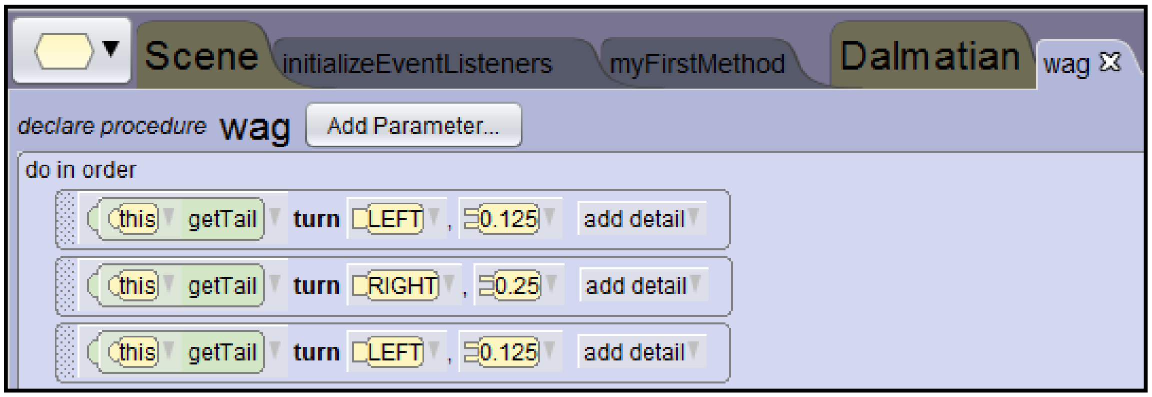
9. Now that the tail subpart for the Dalmatian class selected, let's drag the **turn** method onto the editor for the wag method. Select **LEFT** as the direction argument and **0.125** as the amount argument.



10. Next, let's make the tail wag to the RIGHT. Instead of choosing 0.125 again as our amount, we would need to choose 0.25. The tail has to move double the amount that it moved to the left; otherwise it would end up back at the original starting position. Drag the **turn** method onto the editor choosing a direction argument of **RIGHT** and an amount argument of **0.25**.



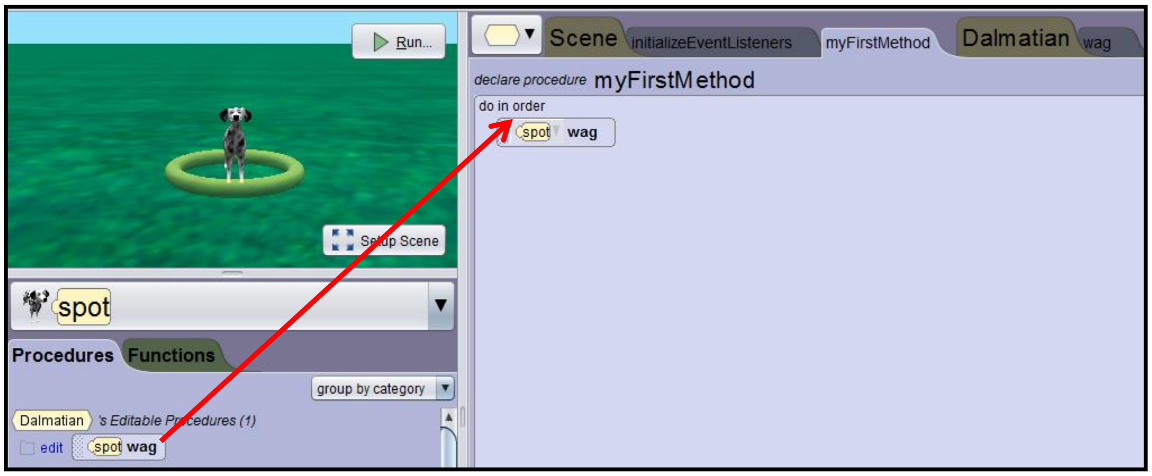
11. Finally, let's add the **turn** method with **LEFT** as the direction argument and **0.125** as the amount argument.



declare procedure wag
do in order
this getTail turn LEFT , 0.125 add detail
this getTail turn RIGHT , 0.25 add detail
this getTail turn LEFT , 0.125 add detail

12. Click the **Run** button to play the animation. You should notice that nothing happens. Although we create a new wag procedural method for all Dalmatian objects, we did not call (invoke) this method.

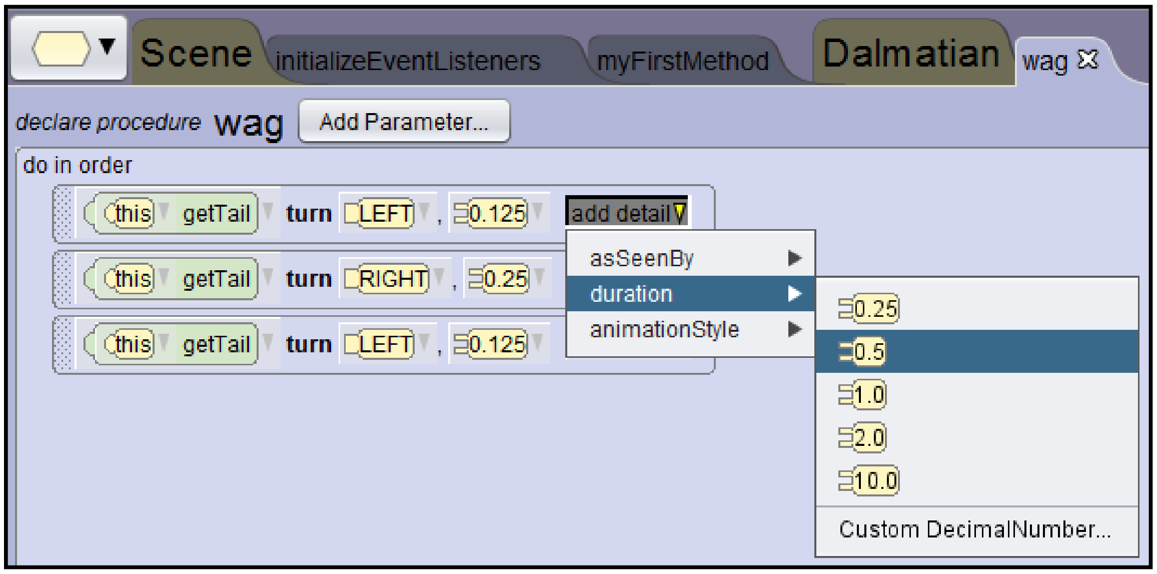
13. Click on the **myFirstMethod** tab. Select **spot** from the object tree. Drag the **wag** method into the editor.



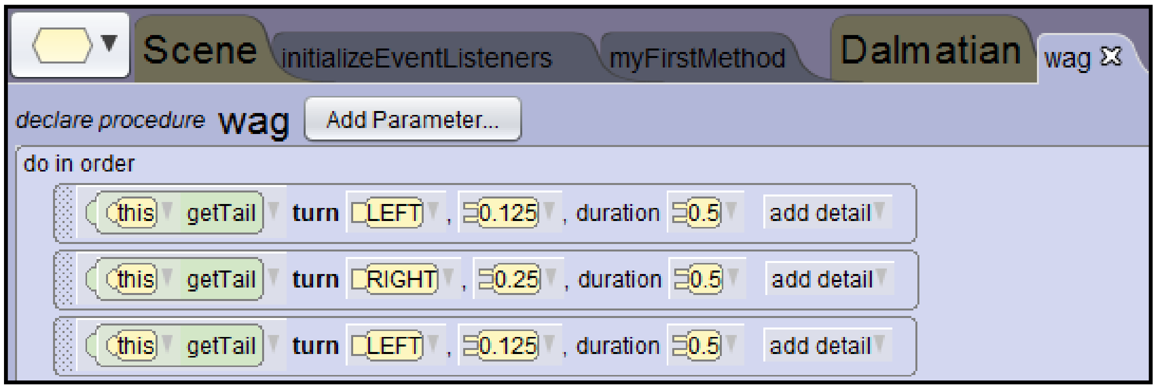
declare procedure myFirstMethod
do in order
spot wag

14. Click the **Run** button to play the animation. The dog should wag its tail.

15. What if we want to have the dog wag its tail at different speeds? Let's click back on the wag method to take a look at our code. Click on **add detail**, then **duration**, select **0.5**.

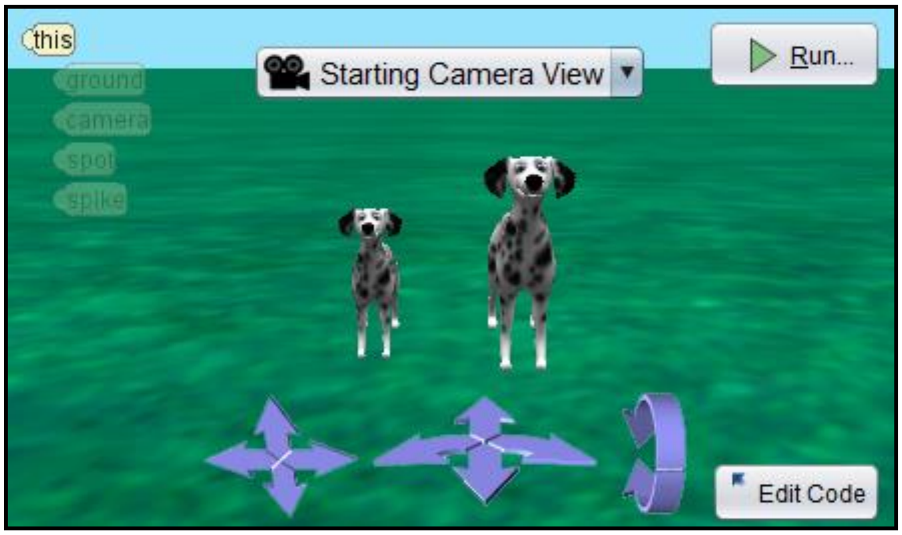


16. Adjust the two turn methods to be 0.5.

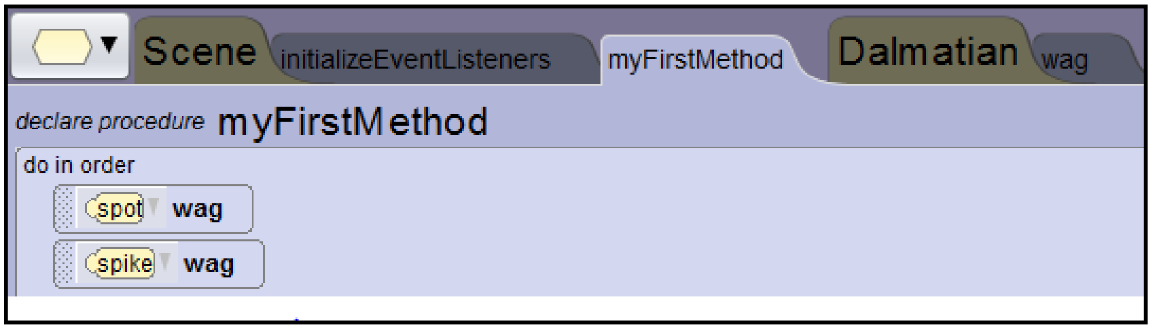


```
declare procedure wag
do in order
  this getTail turn LEFT , 0.125 , duration 0.5  add detail
  this getTail turn RIGHT , 0.25 , duration 0.5  add detail
  this getTail turn LEFT , 0.125 , duration 0.5  add detail
```

17. Click **Setup Scene** button and add another Dalmatian object to your scene. Name this object **spike**.

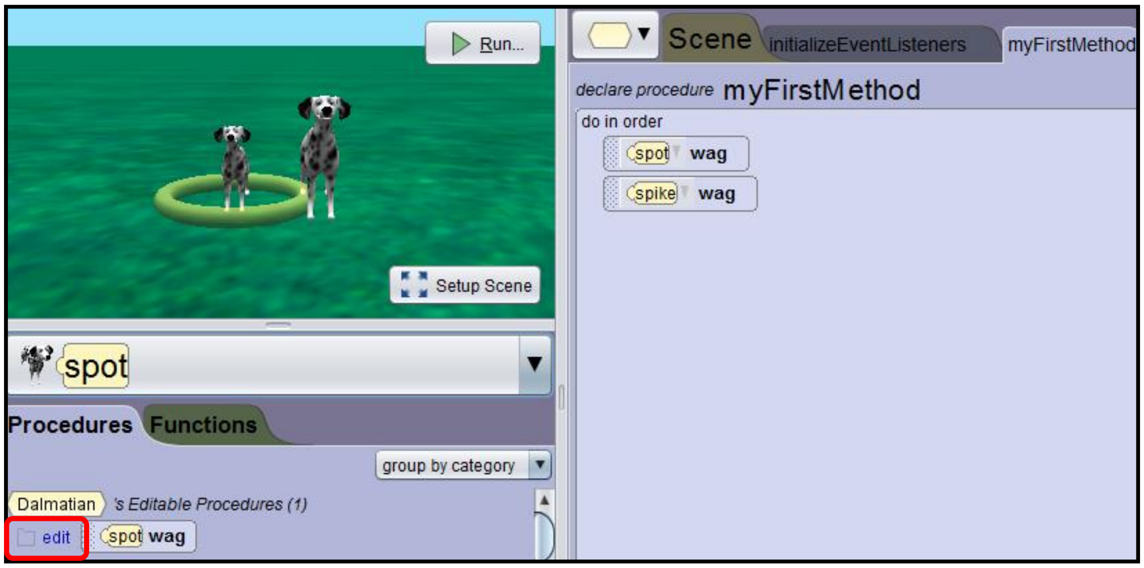


18. Click back on the myFirstMethod tab, select the Dalmatian object **spike**, and drag the **wag** method for spike to the editor. Run your animation. Spot should wag its tail and then spike should wag its tail.



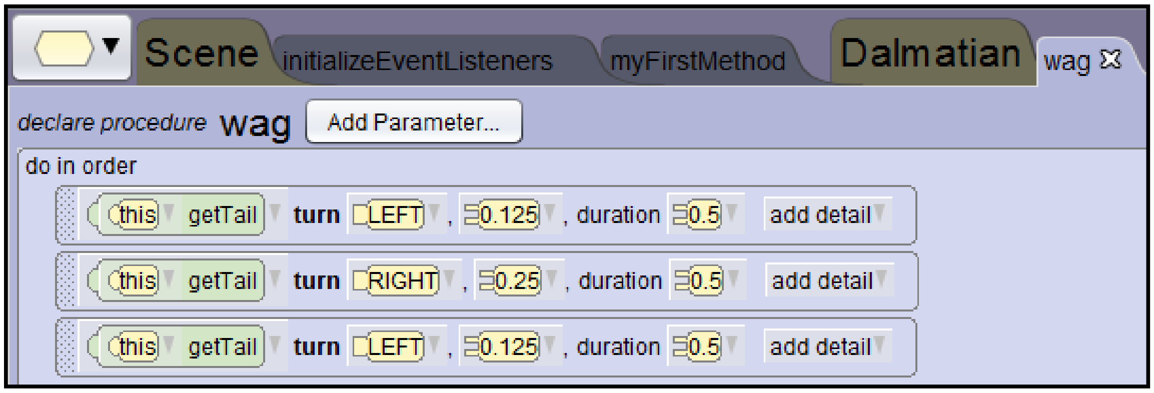
```
declare procedure myFirstMethod
do in order
    spot wag
    spike wag
```

19. If you accidently close the wag method, or closed down the Alice environment and restarted, you will need to reopen the wag method. You can do this by select an object from the Dalmatian class (spot or spike) and clicking edit next to the method name as shown below:



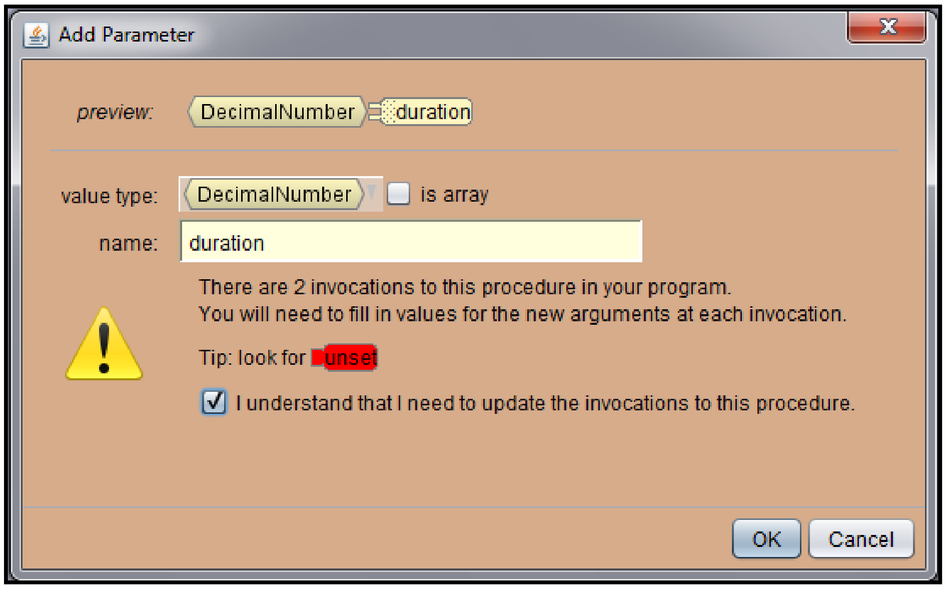
```
declare procedure myFirstMethod
do in order
    spot wag
    spike wag
```

20. What if we wanted to have spot wag his tail at a different speed than spike? The way we have the program currently written, this would not be possible. Click back on the **wag** method and click on the **Add Parameter…** button to add a parameter to our wag method.



```
declare procedure wag
do in order
    this getTail turn LEFT , 0.125 , duration 0.5  add detail
    this getTail turn RIGHT , 0.25 , duration 0.5  add detail
    this getTail turn LEFT , 0.125 , duration 0.5  add detail
```

21. Select **DecimalNumber** as the type *(this is an Alice type, but would be a double in Java)*, name the variable **duration**, and **check the box** for understanding the need for updating invocations to this procedure.



Add Parameter

preview: DecimalNumber duration

value type: DecimalNumber ☐ is array

name: duration

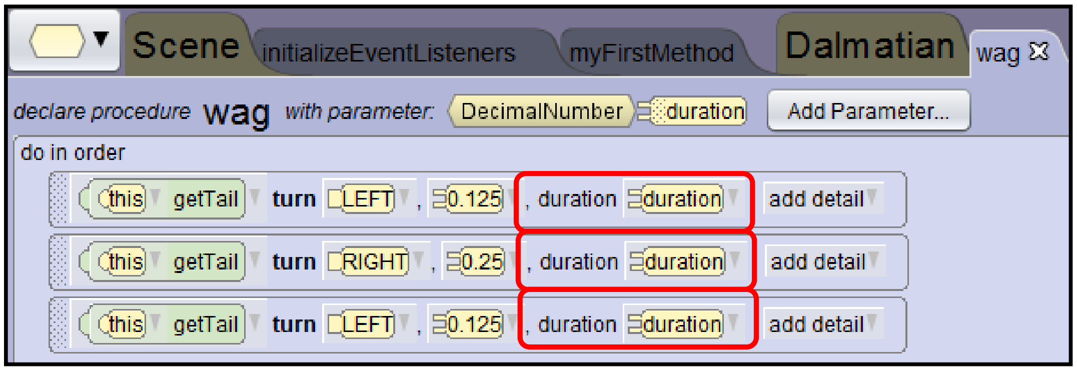
There are 2 invocations to this procedure in your program.
You will need to fill in values for the new arguments at each invocation.

Tip: look for unset

☑ I understand that I need to update the invocations to this procedure.

OK Cancel

22. We will need to put the parameter that we just created into our turn statements. Put the new **duration** parameter as an argument for the duration for each statement.

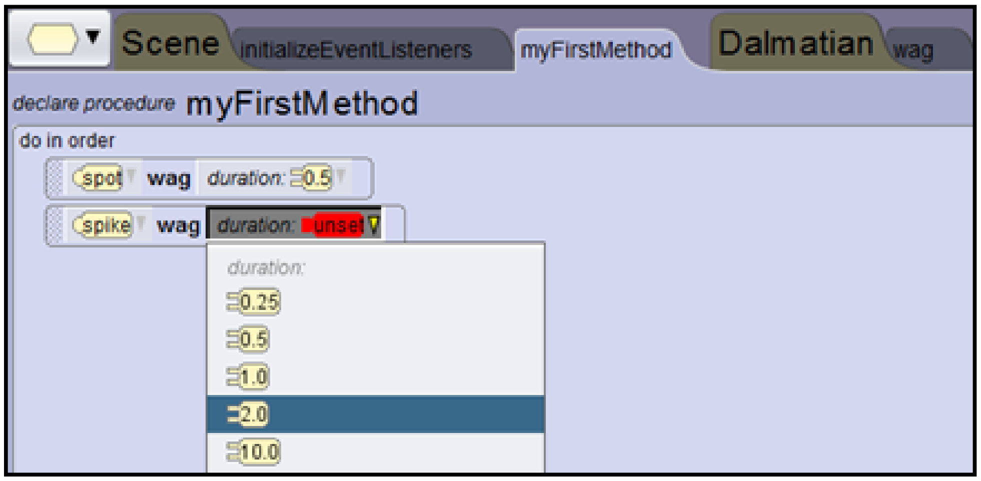


Scene initializeEventListeners myFirstMethod Dalmatian wag

declare procedure wag with parameter: DecimalNumber duration Add Parameter...

```
do in order
    this getTail turn LEFT , 0.125 , duration duration add detail
    this getTail turn RIGHT , 0.25 , duration duration add detail
    this getTail turn LEFT , 0.125 , duration duration add detail
```

23. Now, we need to change the invocations to the wag procedure. Click back on **myFirstMethod** tab and pick a different duration for the wag for each dog.



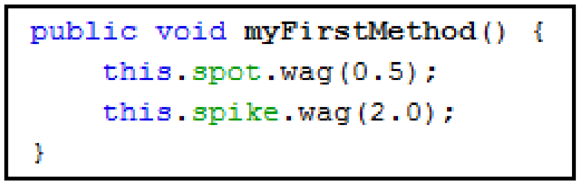
24. Run the animation. Spot and Spike are both able to do the wag method since they are both Dalmatians and the wag method was written for the Dalmatian class. Any Dalmatian object that you add to your scene will be able to wag their tail.

25. Save and exit Alice.

1. Now, let's transfer our Alice project into NetBeans and see what the Java code looks like. Open NetBeans. Click **File**, **New Project**, and select **Java Project from Existing Alice Project** and click the **Next** button.

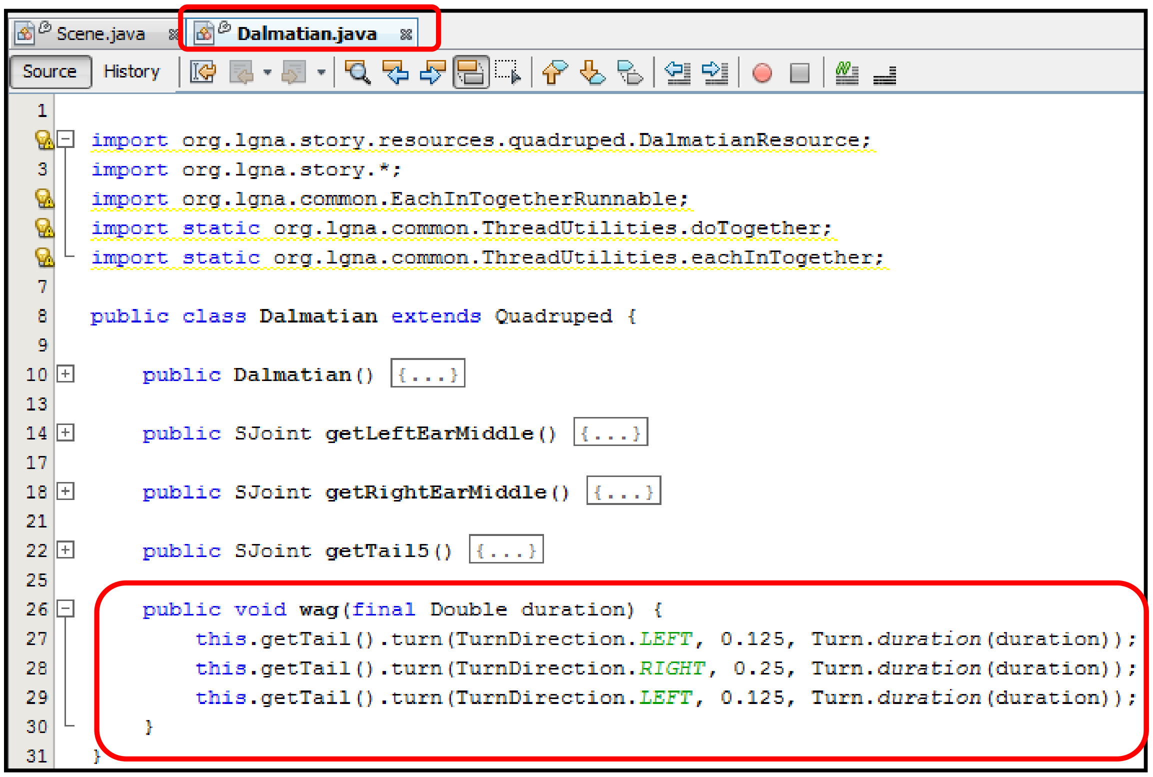
2. Browse for the **DogWag.a3p** Alice file and then be sure to change the location of where you are saving the new NetBeans project. Click **Finish**.

3. If you look at the Java code for this Alice project, you will notice that **myFirstMethod** in the **Scene.java** file calls the wag method for both dogs.



```
public void myFirstMethod() {
    this.spot.wag(0.5);
    this.spike.wag(2.0);
}
```

4. Double click on the **Dalmatian.java** file in the Projects tab. The wag procedural method is located in the Dalmatian class since the wag method belongs with the Dalmatian. *Note: the some of the methods in this file are collapsed in the screenshot below. You can click the plus and minus sign to the left of the method to collapse or expand the code.*



```
import org.lgna.story.resources.quadruped.DalmatianResource;
import org.lgna.story.*;
import org.lgna.common.EachInTogetherRunnable;
import static org.lgna.common.ThreadUtilities.doTogether;
import static org.lgna.common.ThreadUtilities.eachInTogether;

public class Dalmatian extends Quadruped {

    public Dalmatian() {...}

    public SJoint getLeftEarMiddle() {...}

    public SJoint getRightEarMiddle() {...}

    public SJoint getTail5() {...}

    public void wag(final Double duration) {
        this.getTail().turn(TurnDirection.LEFT, 0.125, Turn.duration(duration));
        this.getTail().turn(TurnDirection.RIGHT, 0.25, Turn.duration(duration));
        this.getTail().turn(TurnDirection.LEFT, 0.125, Turn.duration(duration));
    }
}
```

5. Run your project and then close it.

Creating a Hokey Pokey Method in Alice

1. Get into **Alice3**.

2. Choose the **Grass Template** and then click on the **Setup Scene** button.

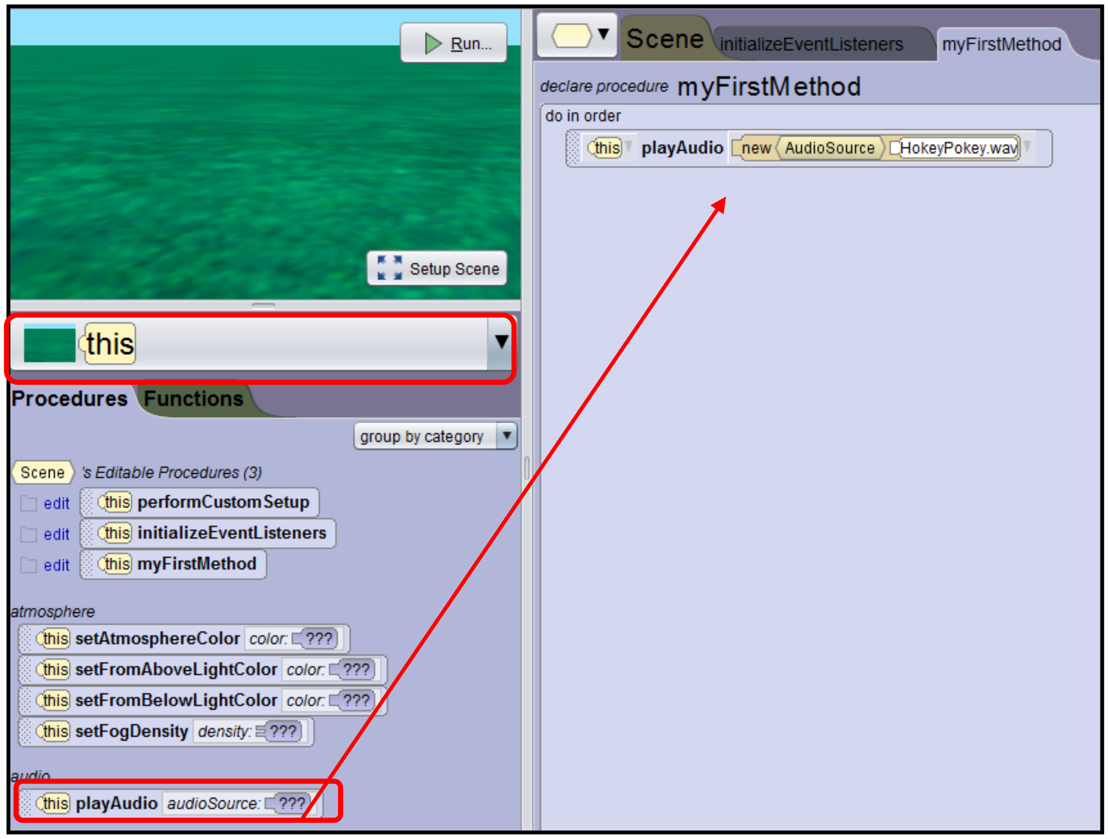
3. Add a tortoise to your world from the biped folder. Name the object **tortoise**.

4. Resize the tortoise so that you can clearly see him. Go back to the code editor.

5. Name your project **HokeyPokey**.

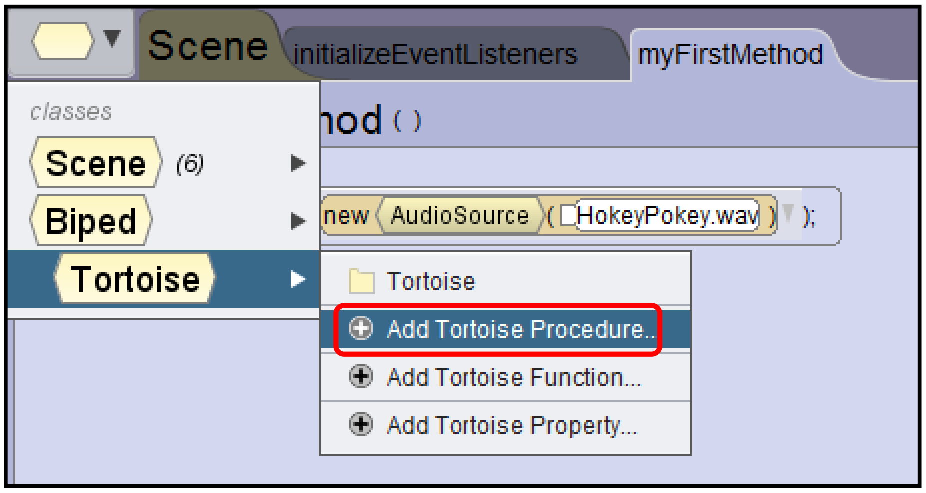
6. We are going to have this tortoise do the hokey pokey. Let's add the Hockey Pokey song so that our tortoise could have some music. Please download the HokeyPokey.wav file and place in same folder where you are placing your Alice projects.

7. Choose the Scene by clicking on **.this**. Scroll down to the **playAudio** procedure (method). Drag this method to myFirstMethod and then choose **Import Audio**. Then find the **HokeyPokey.wav** file. Your statement should look as follows:


Run...
Setup Scene
this
Procedures
Functions
group by category
Scene 's Editable Procedures (3)
edit this performCustomSetup
edit this initializeEventListeners
edit this myFirstMethod
atmosphere
this setAtmosphereColor color: ???
this setFromAboveLightColor color: ???
this setFromBelowLightColor color: ???
this setFogDensity density: ???
audio
this playAudio audioSource: ???
Scene
initializeEventListeners
myFirstMethod
declare procedure myFirstMethod
do in order

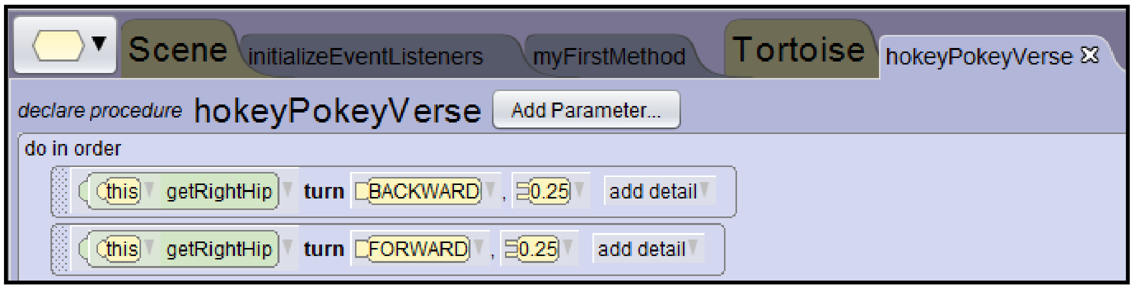
this playAudio new AudioSource HokeyPokey.wav

8. Instead of putting all of our code in our run method as we have been doing in previous chapters, we are going to break the code down into separate methods. The first method that we are going to create is going to be the hokeyPokeyVerse. Method names always start with a lowercase letter. Method naming follows the same rules as variable naming. To create this new procedure method for the tortoise, we will click on the **Tortoise** class and then **Add Tortoise Procedure…**



9. Name the new procedure method **hokeyPokeyVerse**.

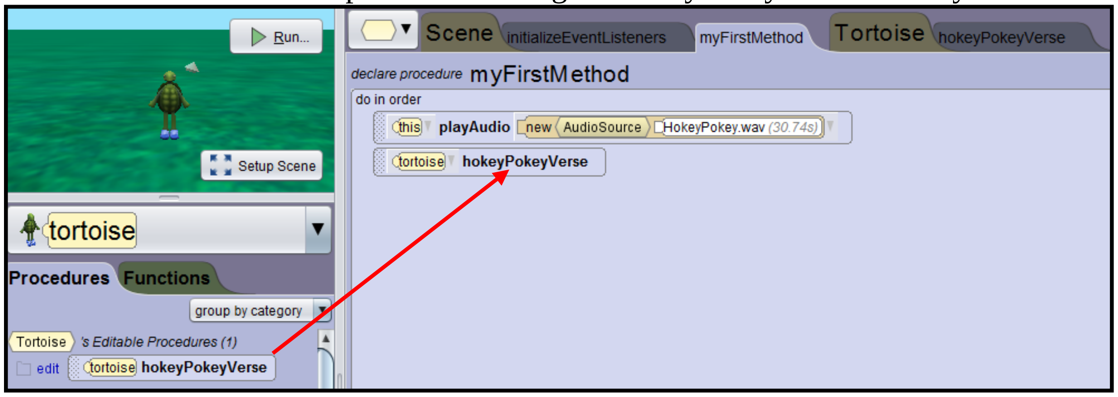
10. We will add the statements to have the tortoise put his right hip in and right hip out. Select the **RightHip** body part. Choose the **turn** method and then fill in the **BACKWARD** and **0.25** as arguments. Add a turn with **FORWARD** and **0.25** as arguments. *(Note: you can copy the first turn method and make adjustments by holding down the control key, command key on the Mac, dragging it down and releasing or by using the clipboard.)*



```
declare procedure hokeyPokeyVerse
do in order
    (this getRightHip turn BACKWARD , 0.25 add detail
    (this getRightHip turn FORWARD , 0.25 add detail
```

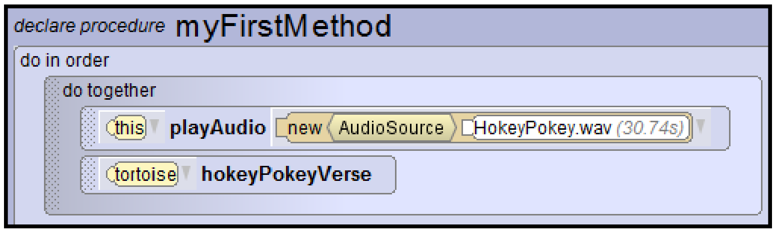
11. If you click on the Run button, you will notice that nothing happens. We need to call the hokeyPokeyVerse procedure method from myFirstMethod. Click on the **myFirstMethod** tab.

12. Click the tortoise from drop down list. Drag the hokeyPokeyVerse onto myFirstMethod.



```
declare procedure myFirstMethod
do in order
    this playAudio new AudioSource HokeyPokey.wav (30.74s)
    tortoise hokeyPokeyVerse
```

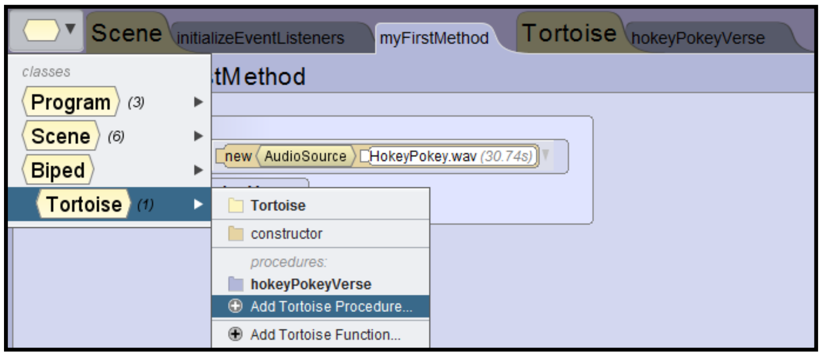
13. Click the **Run** button. What happens? The song is playing to the end and then the tortoise turns his leg. This is a problem. The song and the tortoise hokeyPokeyVerse should be playing at the same time. You can use the **DoTogether** to have the song play at the same time as the leg movements:



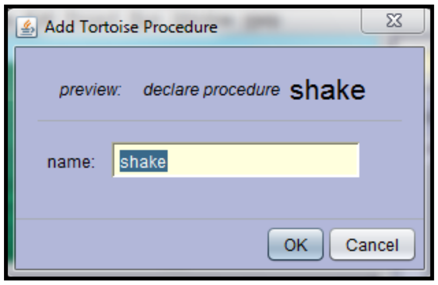
```
declare procedure myFirstMethod
do in order
    do together
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        tortoise hokeyPokeyVerse
```

14. The shake is going to be 6 lines of code. We will have the tortoise turn his right leg to the right, then left, then right, and then left. Instead of adding these 6 lines to our hokeyPokeyVerse method, we want to separate this code into a new procedure method called **shake** that we call from the **hokeyPokeyVerse** method.

15. To create this new procedure method for the tortoise, we will click on the **Tortoise** class, then **Add Tortoise Procedure…**



16. Name the new procedure method **shake**.

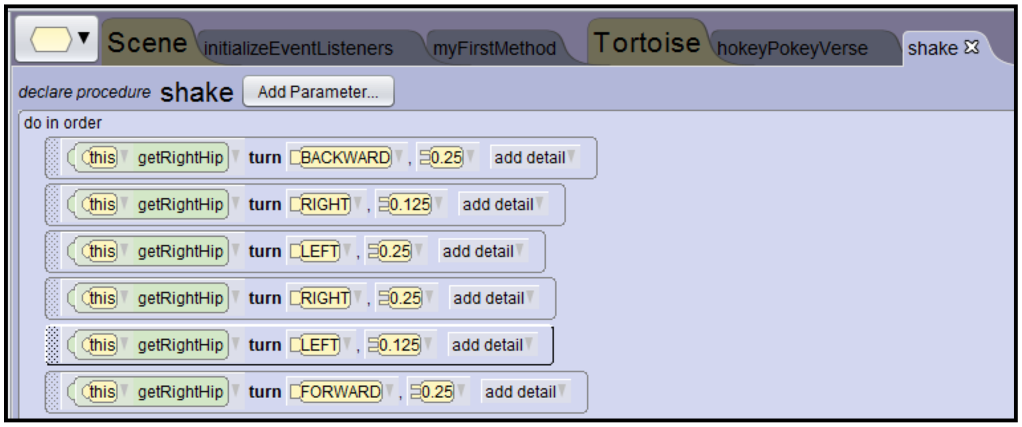


Add Tortoise Procedure

preview: declare procedure shake

name: shake

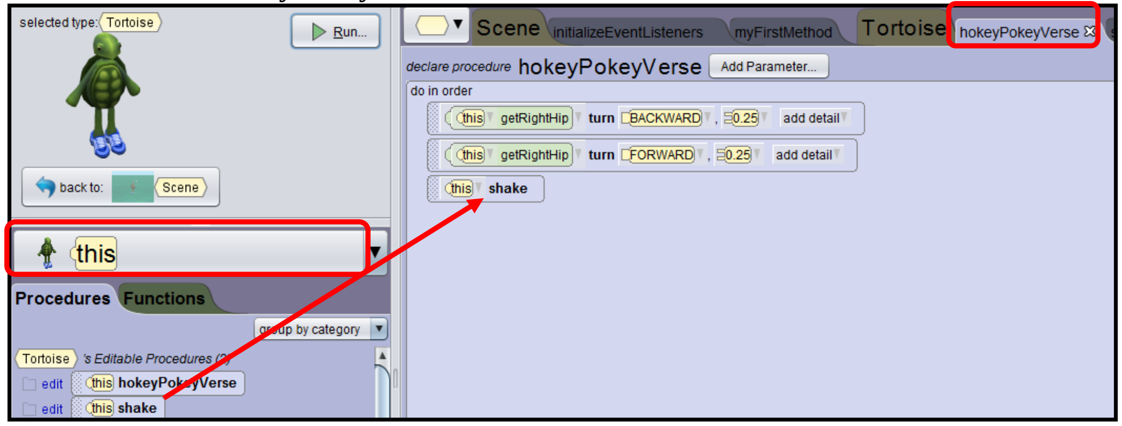
17. We need to add the following lines of code to have the tortoise shake his right hip. We need to turn his right hip backward to get the hip ready for the shake. Then, the right hip should turn 0.125 to the right to begin the shake. To turn his right hip to the left, the tortoise will need to turn his hip 0.25 to right to make up for the 0.125 that his hip has already turned to the right. Then, we repeat this again for the right and left motions except that the left turn should only be 0.125 so that it ends up back at its original position. After, the shaking, we will put the hip back to standing position by turning it forward. Please see the code below.



```
declare procedure shake
do in order
    this getRightHip turn BACKWARD , 0.25 add detail
    this getRightHip turn RIGHT , 0.125 add detail
    this getRightHip turn LEFT , 0.25 add detail
    this getRightHip turn RIGHT , 0.25 add detail
    this getRightHip turn LEFT , 0.125 add detail
    this getRightHip turn FORWARD , 0.25 add detail
```

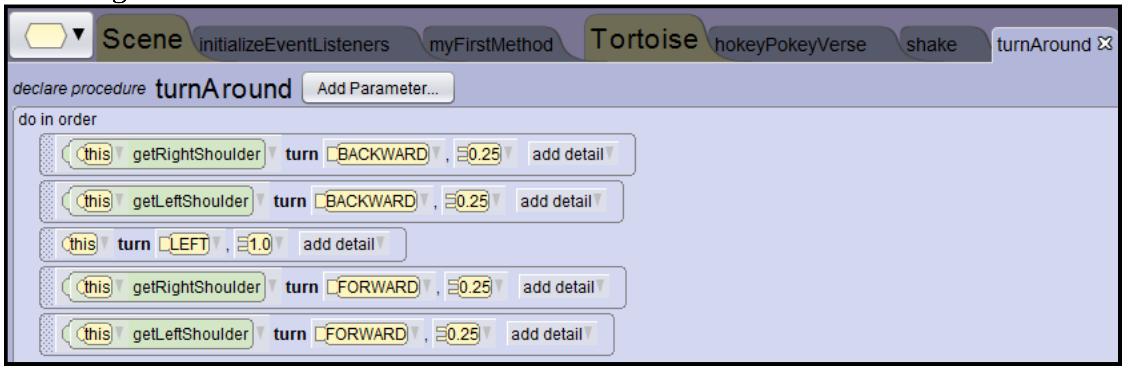
18. Before playing your animation, you need to call the new shake method from the hokeyPokeyVerse method. To get to the hokeyPokey verse method, click on the **hokeyPokeyVerse** tab.

19. Then, click on **this** *(this refers to the current object which is the tortoise)*. Drag the **shake** method into the hokeyPokeyVerse method as shown below:



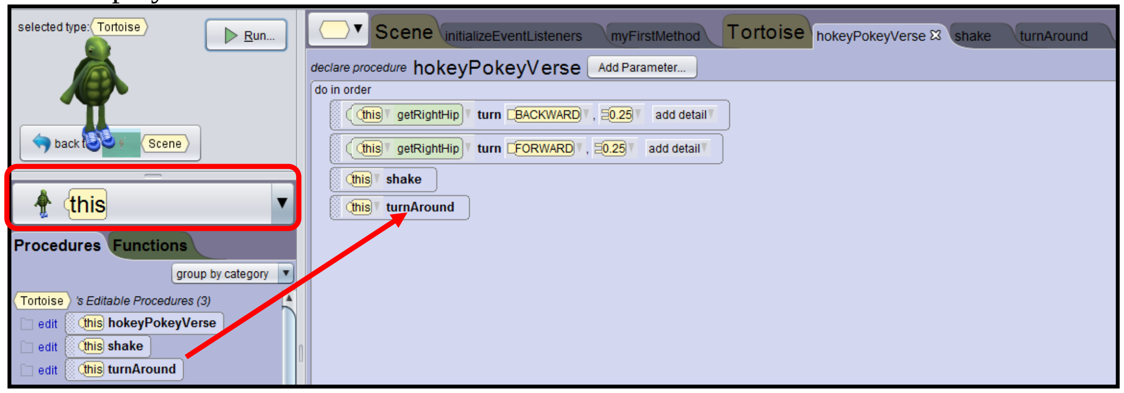
```
declare procedure hokeyPokeyVerse
do in order
    this getRightHip turn BACKWARD , 0.25 add detail
    this getRightHip turn FORWARD , 0.25 add detail
    this shake
```

20. Now we are going to have the tortoise put his arms up in the air and turn around. We will create a new method named **turnAround**. Click on the **Tortoise** class, then **Add Tortoise Procedure…**Give the new procedure method the name **turnAround**. Add the following code for the new turnAround method.



```
declare procedure turnAround
do in order
    this getRightShoulder turn BACKWARD , 0.25 add detail
    this getLeftShoulder turn BACKWARD , 0.25 add detail
    this turn LEFT , 1.0 add detail
    this getRightShoulder turn FORWARD , 0.25 add detail
    this getLeftShoulder turn FORWARD , 0.25 add detail
```

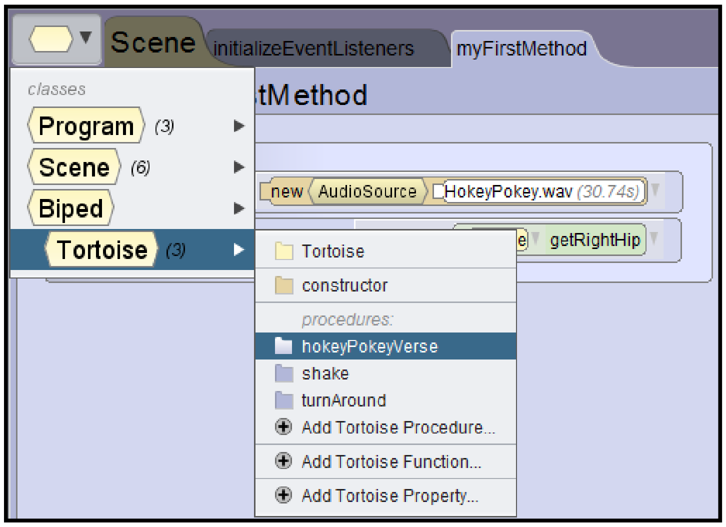
21. Now, we need to call the turnAround method from the hokeyPokeyVerse method in order for it to play.



```
declare procedure hokeyPokeyVerse
do in order
    this getRightHip turn BACKWARD , 0.25 add detail
    this getRightHip turn FORWARD , 0.25 add detail
    this shake
    this turnAround
```

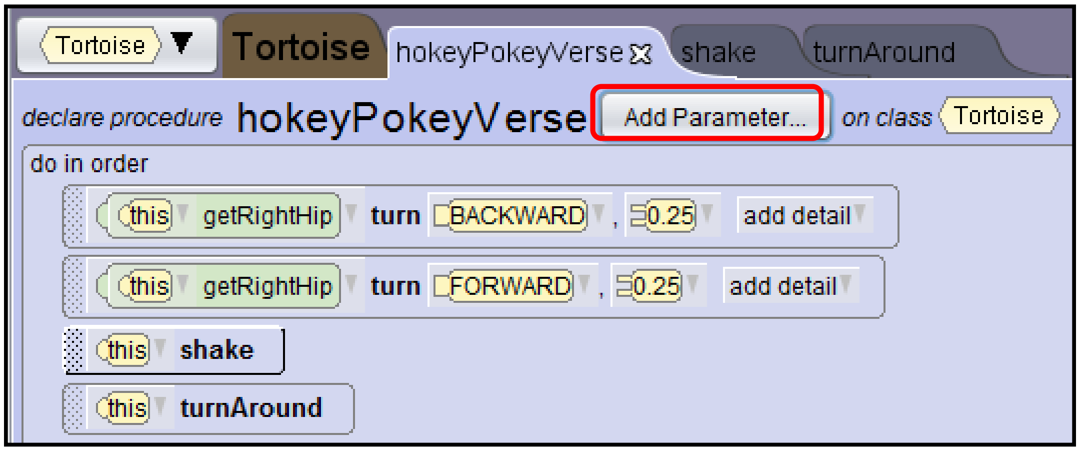
22. Play your animation. This looks pretty good so far, but wouldn't it be nice if this worked for the left leg, right arm, etc.? You are half way done with this exercise. This is a good stopping point if you want to take a break. Please save and exit Alice.

23. **Continue:** If you took a break, please open Alice, open your HokeyPokey program, and open all your method tabs for the Hokey Pokey *(click on the Tortoise class and then click on each method to open the tab for that method)*.



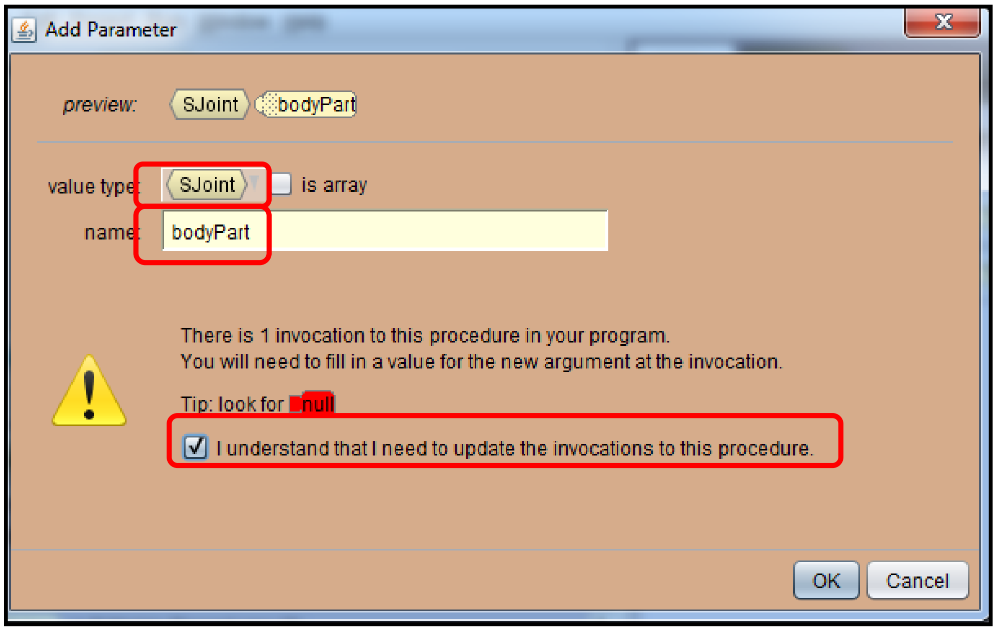
24. We already programmed the tortoise to do the hokey pokey with his right leg, but if we wanted to switch body parts, we shouldn't have to rewrite the same methods over and over again for every body part. Since the only thing that will be changing will be the body part, we can set up a parameter to pass the body part into the methods that we already created. Let's click on the **hokeyPokeyVerse** method tab to open up the code for this method.

25. We are going to add a parameter to this method. You can do this by clicking on the **Add Parameter…** button.



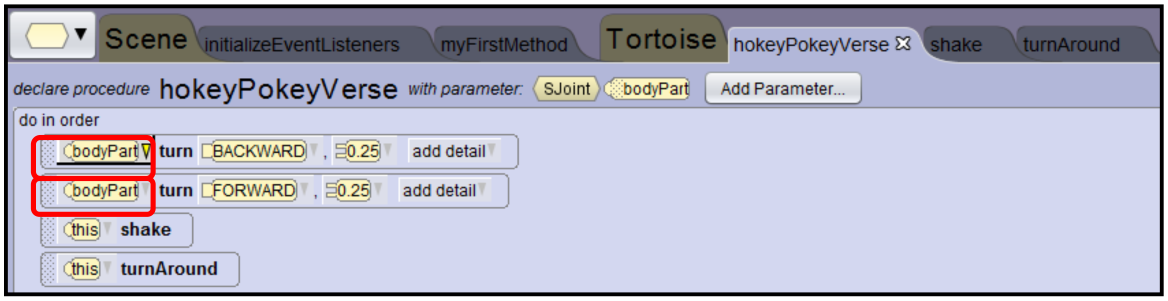
```
declare procedure hokeyPokeyVerse [Add Parameter...] on class Tortoise
do in order
    (this) getRightHip turn BACKWARD , 0.25 add detail
    (this) getRightHip turn FORWARD , 0.25 add detail
    this shake
    this turnAround
```

26. This parameter should have a type of **Other Types… SJoint**, name it **bodyPart**, and check the box that reads **I understand….**



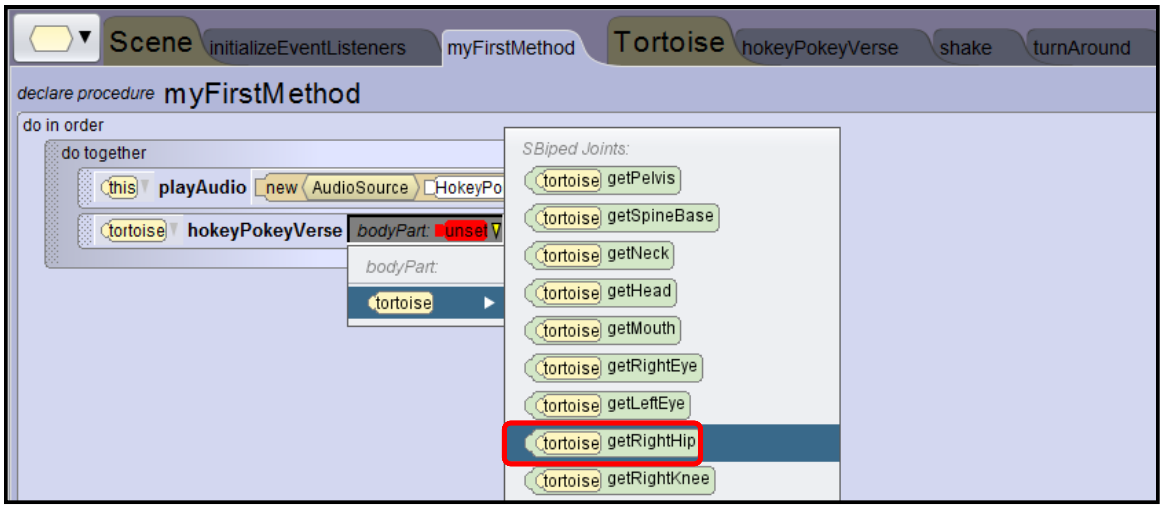
There is 1 invocation to this procedure in your program.
You will need to fill in a value for the new argument at the invocation.
Tip: look for null
☑ I understand that I need to update the invocations to this procedure.

27. Now we are going to replace the right hip with the new bodyPart variable. Drag the bodyPart parameter onto the **this.getRightHip** as shown below. You can also click the drop down next to this.getRightHip to select the bodyPart variable.



```
declare procedure hokeyPokeyVerse with parameter: SJoint bodyPart
do in order
    bodyPart turn BACKWARD , 0.25 add detail
    bodyPart turn FORWARD , 0.25 add detail
    this shake
    this turnAround
```

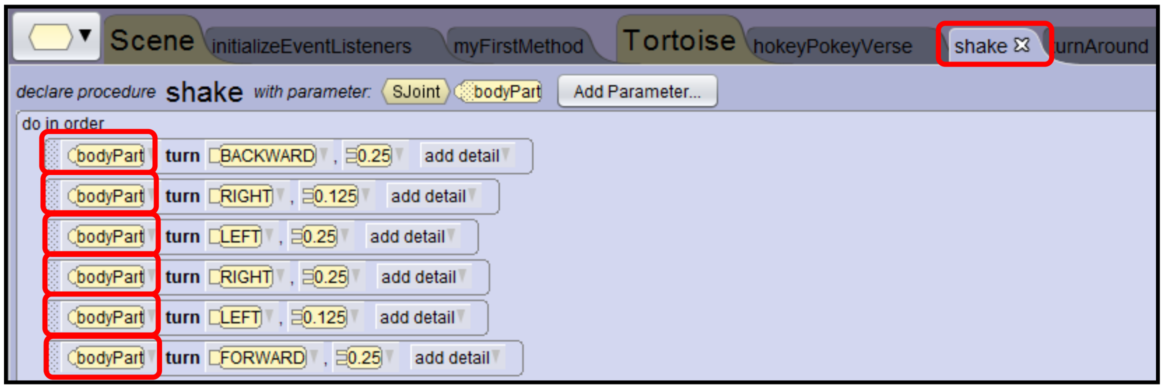
28. Now we have to go back to **myFirstMethod**. We need to pass a bodyPart as an argument to the hokeyPokeyVerse method. Click the drop-down next to hokeyPokeyVerse and choose the getRightHip. If you do not change the bodyPart from unset to a body part, Alice will crash *(it cannot do the verse without a body part).*



29. Play your animation. It should still look the same since we are using the right leg.

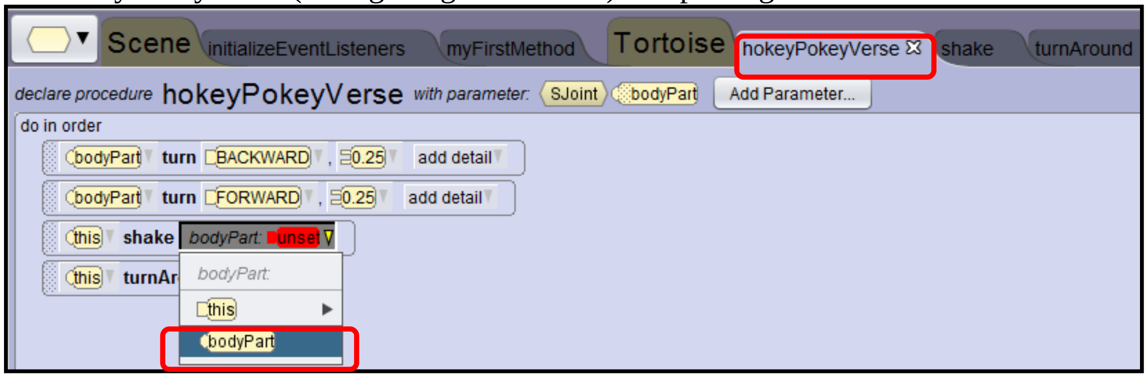
30. Now, we need to fix the shake method the same way. We should add a parameter and we can name it bodyPart. Go to the **shake** method tab. We are going to add a parameter to this method. You can do this by clicking on the **Add Parameter…** button. This parameter should be type of **OtherTypes…. SJoint,** name it **bodyPart**, and check the **I understand….** box.

31. We are going to replace the **this.getRightLeg** with the bodyPart (parameter) as shown below.



```
declare procedure shake with parameter: SJoint bodyPart
do in order
    bodyPart turn BACKWARD , 0.25 add detail
    bodyPart turn RIGHT , 0.125 add detail
    bodyPart turn LEFT , 0.25 add detail
    bodyPart turn RIGHT , 0.25 add detail
    bodyPart turn LEFT , 0.125 add detail
    bodyPart turn FORWARD , 0.25 add detail
```

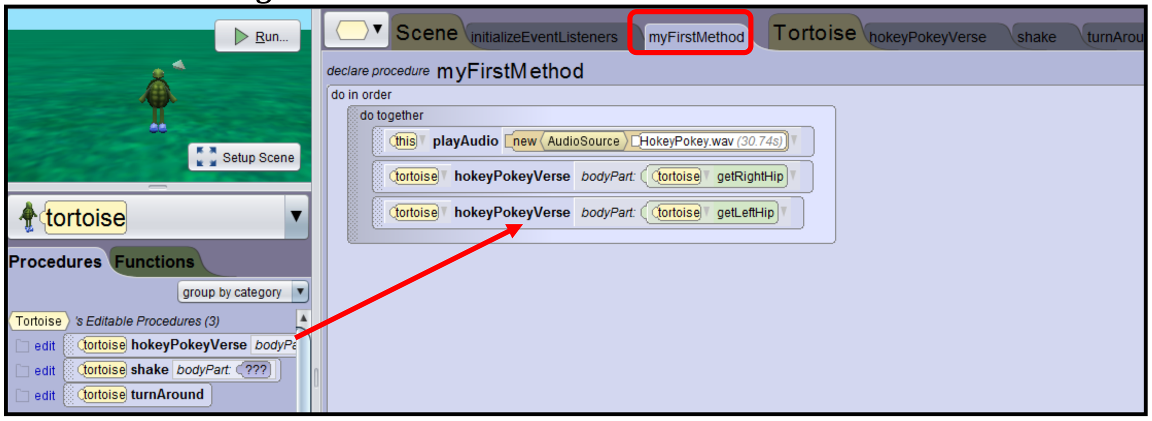
32. Now we need to go back to the place where we called this method. Click on the **hokeyPokeyVerse** method tab. Select bodyPart from the drop-down next to the shake method bodyPart parameter. We are actually using the **bodyPart** that was passed in to the hokeyPokeyVerse (the right leg in this case) and passing that into our shake method.



```
declare procedure hokeyPokeyVerse with parameter: SJoint bodyPart
do in order
    bodyPart turn BACKWARD , 0.25 add detail
    bodyPart turn FORWARD , 0.25 add detail
    this shake bodyPart: unset
    this turnAr
```

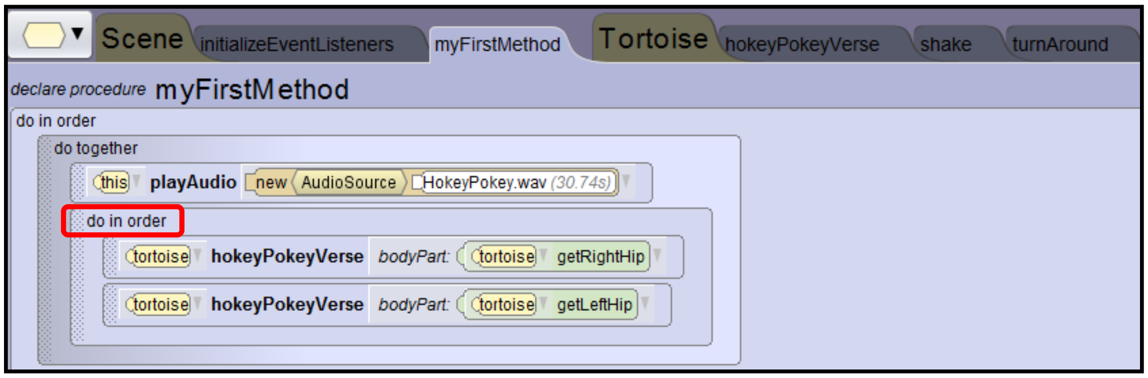
33. Now play your animation. It should still work for the right leg. We don't need to worry about adding a parameter for the turnAround method since the tortoise will be raising his arms and turning around exactly the same way each time.

34. Let's go back to the **myFirstMethod** and call the **hokeyPokeyVerse** method again this time with the **left leg**.



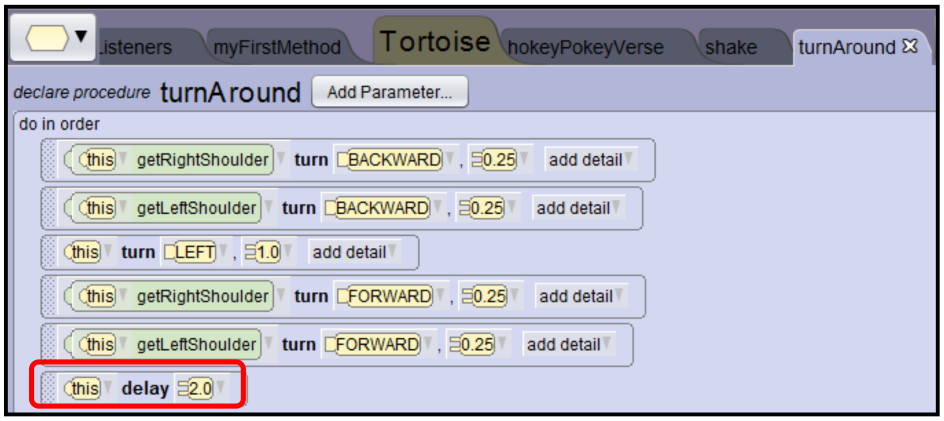
```
declare procedure myFirstMethod
do in order
    do together
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        tortoise hokeyPokeyVerse bodyPart: tortoise getRightHip
        tortoise hokeyPokeyVerse bodyPart: tortoise getLeftHip
```

35. You will notice that the right hip and left hip move at the same time. You will need to add a DoInOrder to fix this issue as shown below:



```
declare procedure myFirstMethod
do in order
    do together
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        do in order
            tortoise hokeyPokeyVerse bodyPart: tortoise getRightHip
            tortoise hokeyPokeyVerse bodyPart: tortoise getLeftHip
```

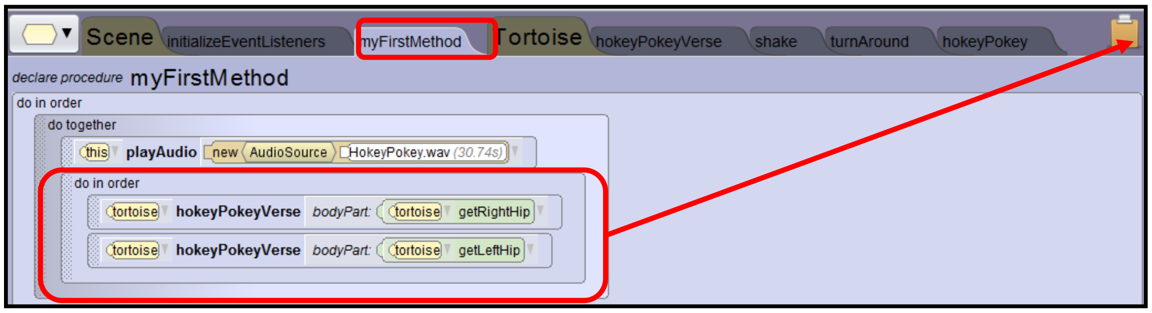
36. Everything works fine except the timing with the song may be off by 2 seconds. We need a delay at the end of the turnAround so that it doesn't start the next leg too soon. Click on the turnAround method and adjust it to have the delay of 2 seconds.



```
declare procedure turnAround
do in order
  this getRightShoulder turn BACKWARD , 0.25 add detail
  this getLeftShoulder turn BACKWARD , 0.25 add detail
  this turn LEFT , 1.0 add detail
  this getRightShoulder turn FORWARD , 0.25 add detail
  this getLeftShoulder turn FORWARD , 0.25 add detail
  this delay 2.0
```

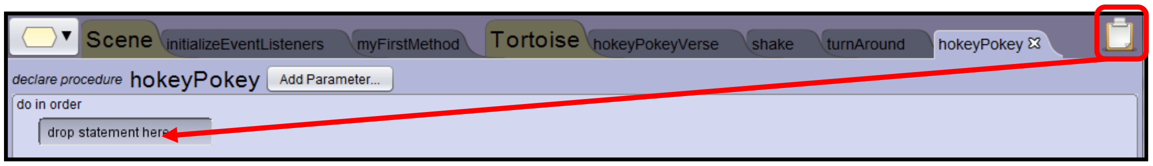
37. We need to have one method called hokeyPokey which calls the hokeyPokeyVerse which calls the shake and turnaround methods. Create the **hokeyPokey** method for the Tortoise.

38. Drag the DO IN ORDER block from myFirstMethod onto the clipboard *(if you hold down the ctrl key while you drag to the clip board it will copy and if you just drag to the clip board it will cut)*. Since we want to remove this code from myFirstMethod, we should use cut instead of copy.



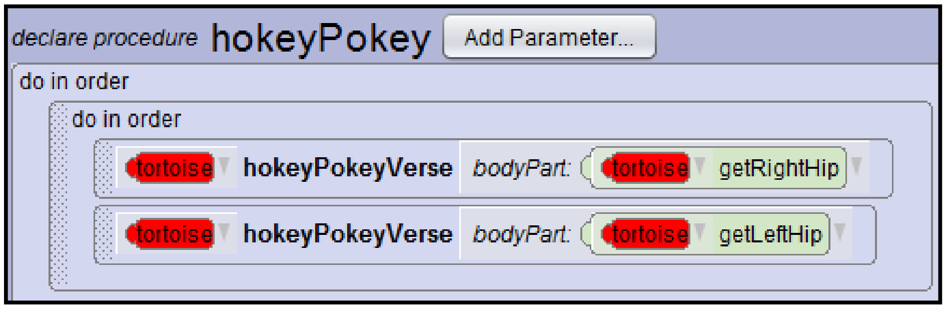
```
declare procedure myFirstMethod
do in order
  do together
    this playAudio new AudioSource HokeyPokey.wav (30.74s)
    do in order
      tortoise hokeyPokeyVerse bodyPart: tortoise getRightHip
      tortoise hokeyPokeyVerse bodyPart: tortoise getLeftHip
```

39. Click on the hokeyPokey method tab and drag the code from the clipboard to the hokeyPokey method *(the white piece of paper on the clipboard represents your code)*.

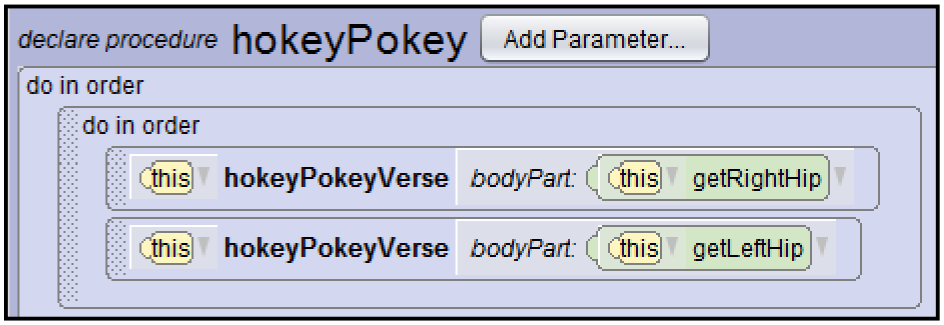


```
declare procedure hokeyPokey
do in order
  drop statement here
```

40. DO NOT run your program yet. You will notice some red in you the code that you pasted *(red indicates errors in code)*. You still have a reference to a particular tortoise and it should be referencing the current object (this) instead of a particular object. Change the reference to the tortoise object named “tortoise” to be the word **this** to represent any tortoise object. The concept of “this” will make more sense as we progress through the course.

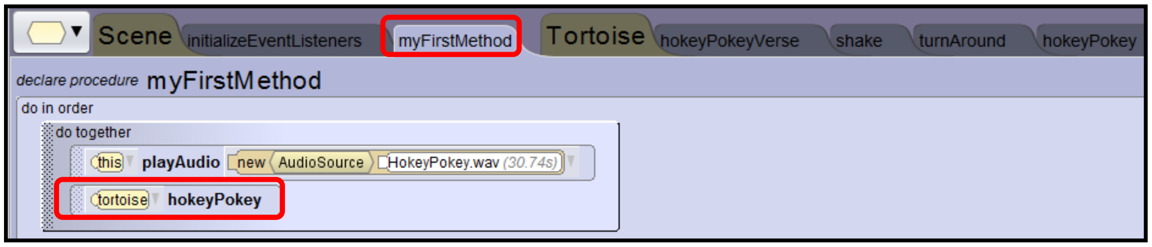


```
declare procedure hokeyPokey  Add Parameter...
do in order
    do in order
        tortoise hokeyPokeyVerse bodyPart: tortoise getRightHip
        tortoise hokeyPokeyVerse bodyPart: tortoise getLeftHip
```



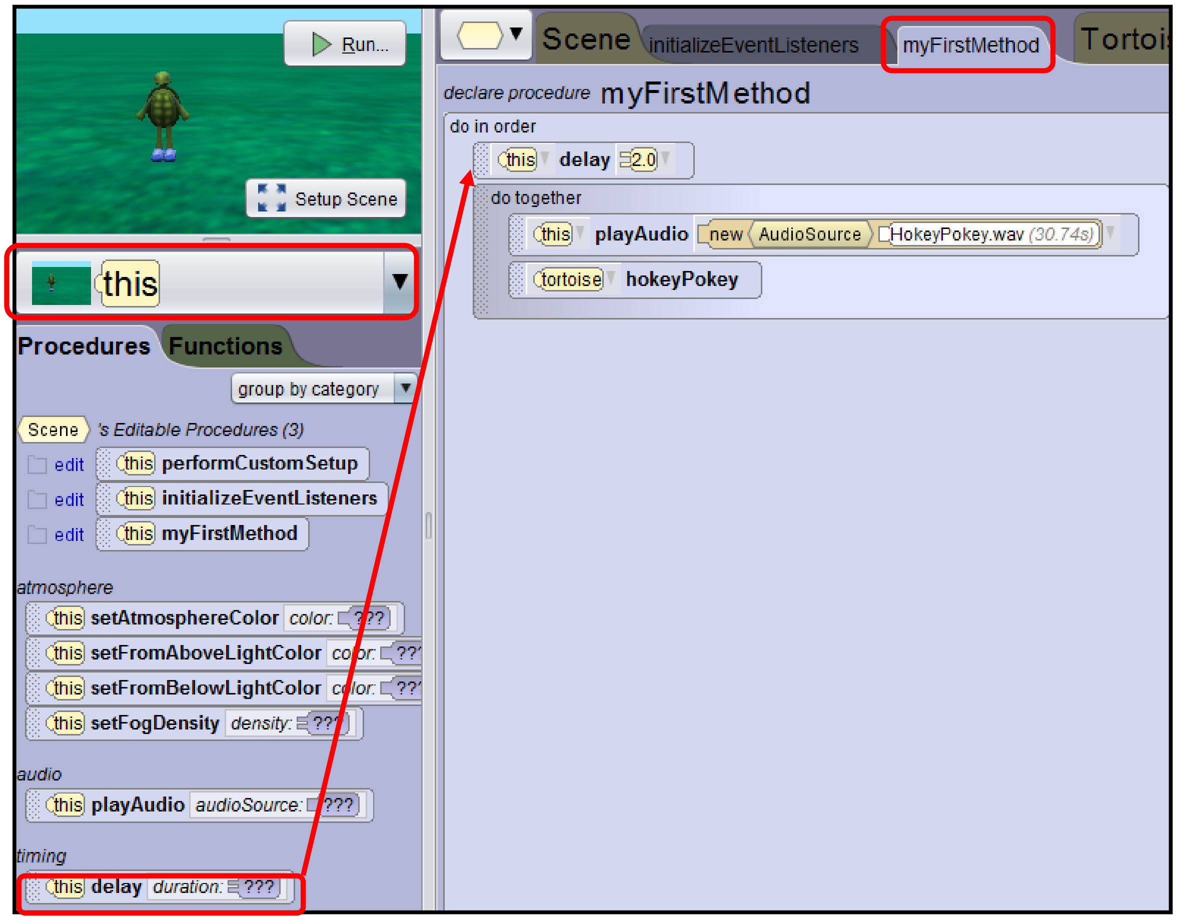
```
declare procedure hokeyPokey  Add Parameter...
do in order
    do in order
        this hokeyPokeyVerse bodyPart: this getRightHip
        this hokeyPokeyVerse bodyPart: this getLeftHip
```

41. Invoke (call) the hokeyPokey method for the tortoise in myFirstMethod *(make sure that you select the tortoise object and click on the procedures tab).*



```
declare procedure myFirstMethod
do in order
    do together
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        tortoise hokeyPokey
```

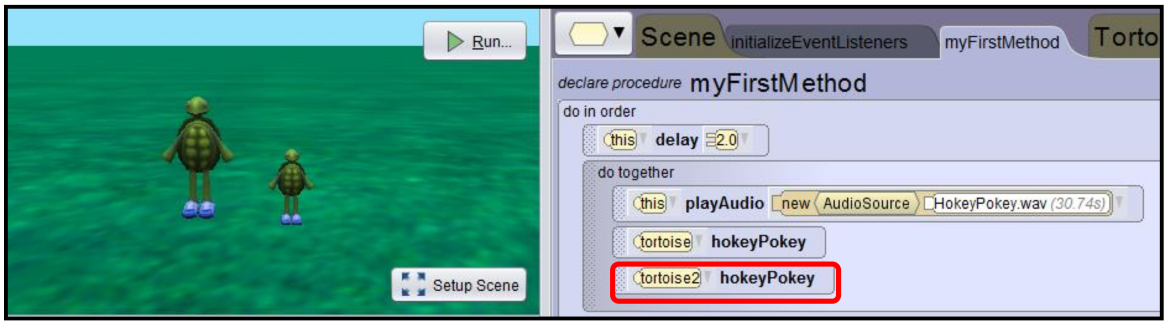
42. You may need to add a delay before your DoTogether block to give the animation a chance to load the sound file. Make sure that you are in myFirstMethod and you have the scene selected “this”. Drag the delay statement above the DoTogether block and choose 2 seconds as the argument.



declare procedure myFirstMethod
do in order
this delay 2.0
do together
this playAudio new AudioSource HokeyPokey.wav (30.74s)
tortoise hokeyPokey

Scene 's Editable Procedures (3)
edit this performCustomSetup
edit this initializeEventListeners
edit this myFirstMethod
atmosphere
this setAtmosphereColor color: ???
this setFromAboveLightColor color: ???
this setFromBelowLightColor color: ???
this setFogDensity density: ???
audio
this playAudio audioSource: ???
timing
this delay duration: ???

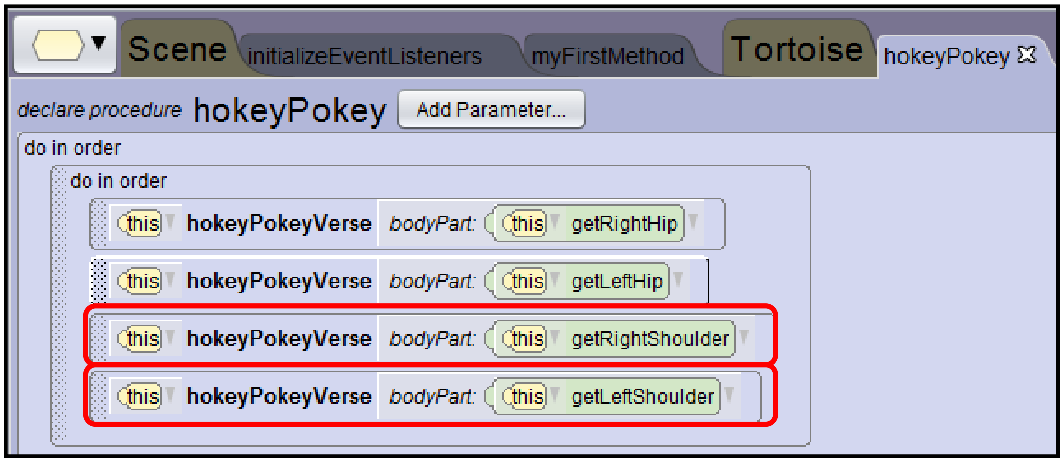
43. Add another tortoise called tortoise2 to the scene. Have that tortoise do the Hokey Pokey also.



declare procedure myFirstMethod
do in order
this delay 2.0
do together
this playAudio new AudioSource HokeyPokey.wav (30.74s)
tortoise hokeyPokey
tortoise2 hokeyPokey

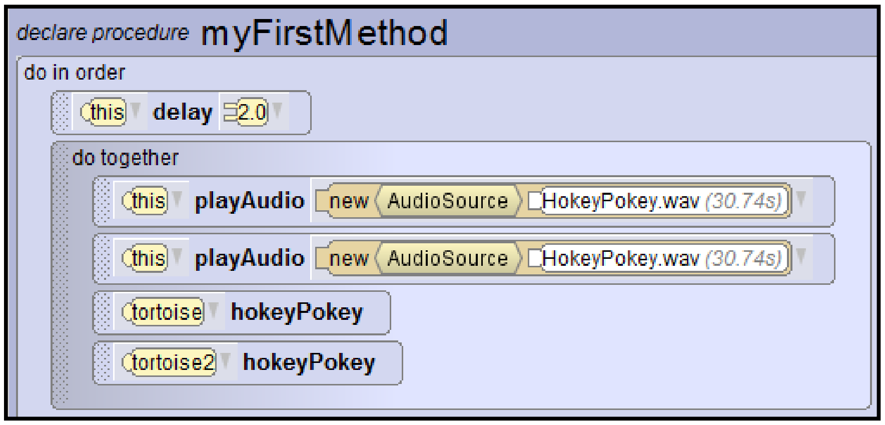
44. Run your animation. Both tortoises should do the Hokey Pokey dance.

45. Let's add the right and left shoulder to the hokeyPokey method as shown below. This verse doesn't work for all of the body parts, since some body parts need to turn forward and some need to turn backward.



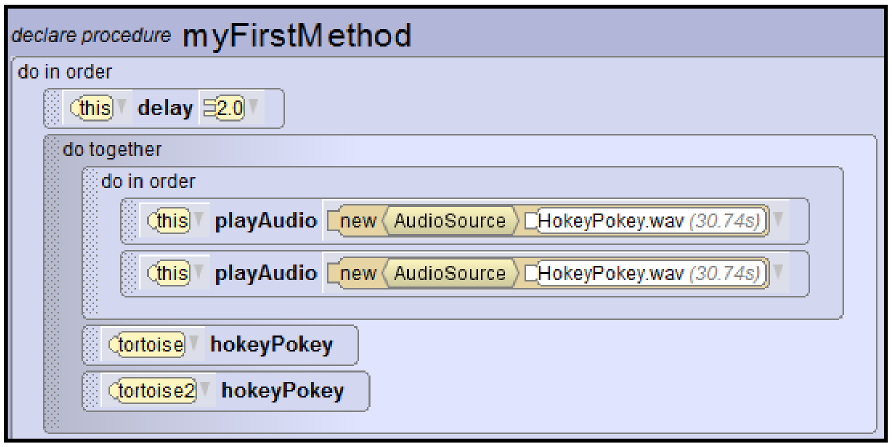
```
declare procedure hokeyPokey  Add Parameter...
do in order
    do in order
        this hokeyPokeyVerse bodyPart: this getRightHip
        this hokeyPokeyVerse bodyPart: this getLeftHip
        this hokeyPokeyVerse bodyPart: this getRightShoulder
        this hokeyPokeyVerse bodyPart: this getLeftShoulder
```

46. Run your animation. You should notice that the song stops after 2 body parts. We need the song to play twice. Copy the playAudio statement so it looks as follows:



```
declare procedure myFirstMethod
do in order
    this delay 2.0
    do together
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        this playAudio new AudioSource HokeyPokey.wav (30.74s)
        tortoise hokeyPokey
        tortoise2 hokeyPokey
```

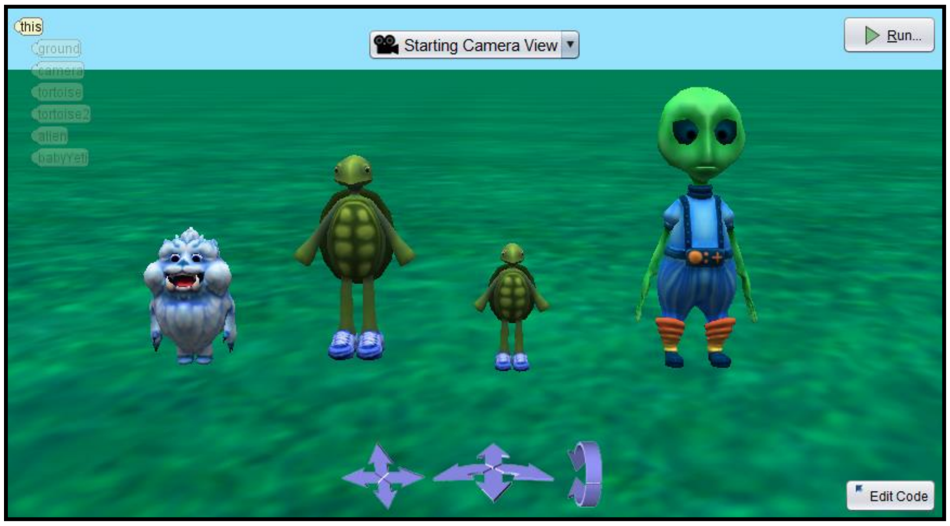
47. Run the animation.You should notice that the song isn't any longer than it was before. Even though we have the line of code twice, it is inside a DoTogether block and so it happens at the same time, therefore canceling the second playAudio line out. We need to put these 2 lines in a **do in order** to keep them from playing at the same time.



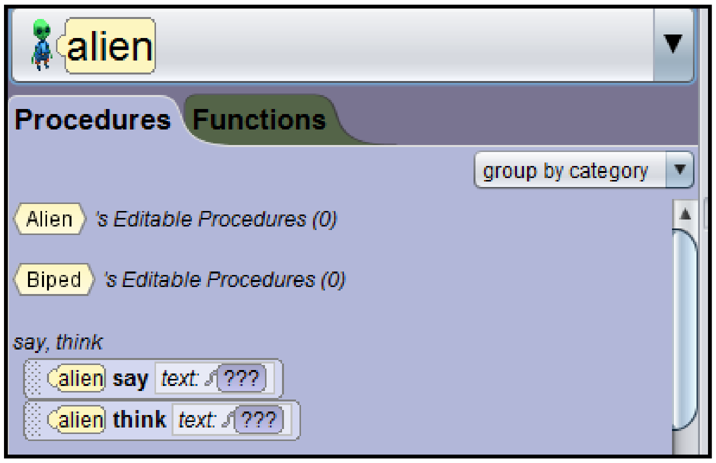
```
declare procedure myFirstMethod
do in order
    this delay 2.0
    do together
        do in order
            this playAudio new AudioSource HokeyPokey.wav (30.74s)
            this playAudio new AudioSource HokeyPokey.wav (30.74s)
        tortoise hokeyPokey
        tortoise2 hokeyPokey
```

48. Run the animation. Looks pretty good. Now we are going to add some more characters to your environment.

49. Add an alien and a baby yeti to the environment to do the Hokey Pokey.



50. Why isn’t there a hokeyPokey method for the alien? The hokeyPokey is a method that belongs to the Tortoise class and therefore the alien and baby yeti do not have access to this method.

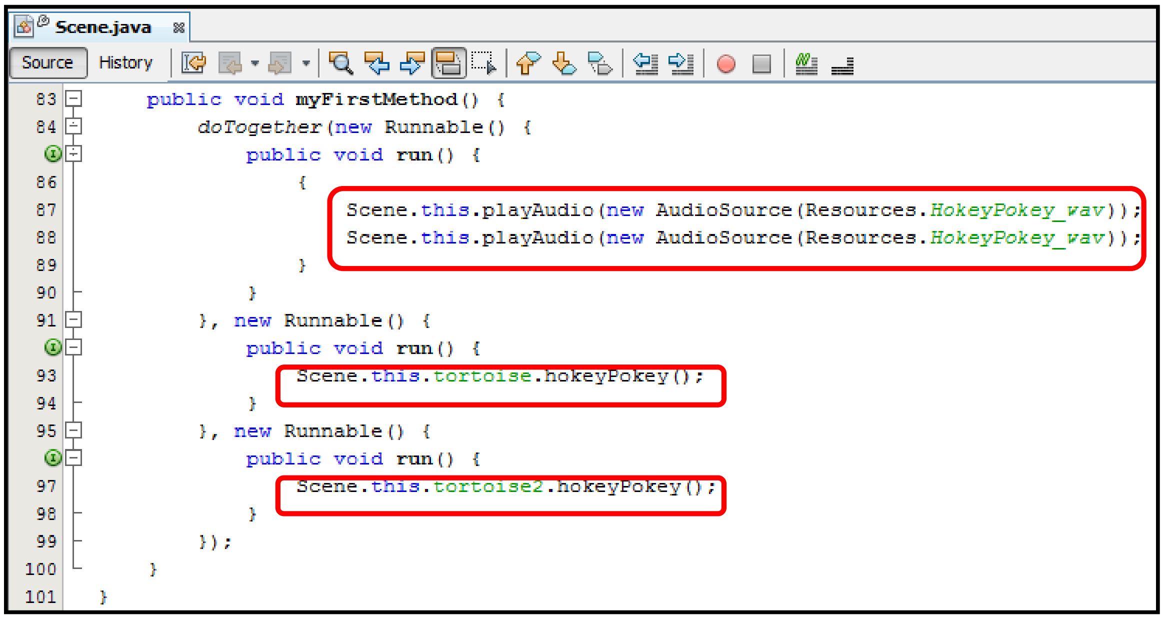


51. Save your Alice project and close the Alice environment.

52. Now, let’s transfer our Alice project into NetBeans and see what the Java code looks like. Open NetBeans. Click **File**, **New Project**, and select **Java Project from Existing Alice Project**. Click the **Next** button.

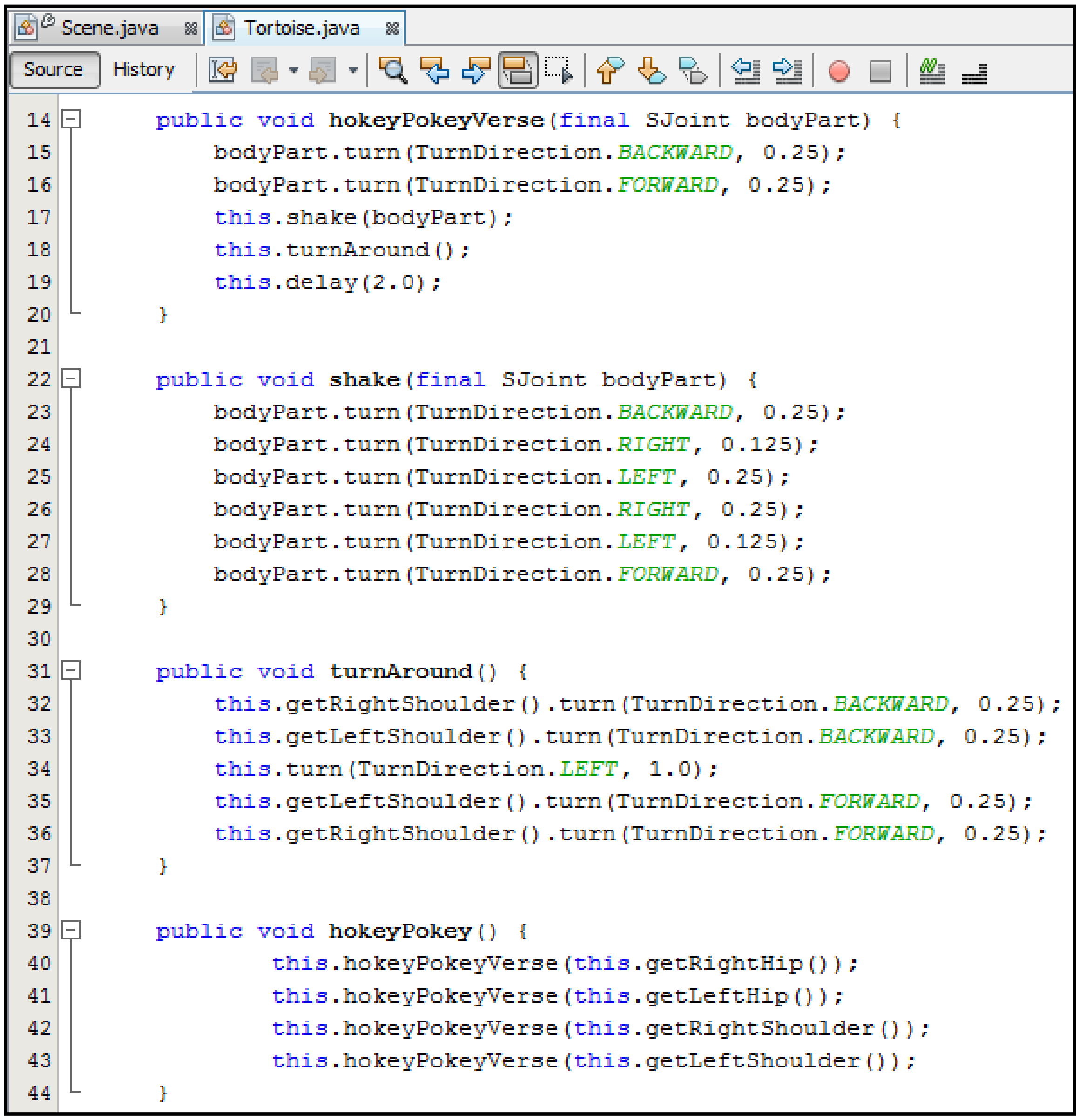
53. Browse for the HokeyPokey Alice file and then be sure to change the location of where you are saving the new NetBeans project. Click **Finish**.

54. Open the **Scene.java** file. The code in myFirstMethod should look familiar. Please ignore the DoTogether code and just focus on the highlight lines.



```
    public void myFirstMethod() {
        doTogether(new Runnable() {
            public void run() {
                {
                    Scene.this.playAudio(new AudioSource(Resources.HokeyPokey_wav));
                    Scene.this.playAudio(new AudioSource(Resources.HokeyPokey_wav));
                }
            }
        }, new Runnable() {
            public void run() {
                Scene.this.tortoise.hokeyPokey();
            }
        }, new Runnable() {
            public void run() {
                Scene.this.tortoise2.hokeyPokey();
            }
        });
    }
}
```

55. Open the **Tortoise.java** file. You should see the hokeyPokeyVerse, shake, turnAround, and newly added hokeyPokey methods. Notice how none of the methods that we created are static? Static methods are methods that are not called on an object and since most of the methods that we will write in Alice will be using objects, they will not be static.



```java
    public void hokeyPokeyVerse(final SJoint bodyPart) {
        bodyPart.turn(TurnDirection.BACKWARD, 0.25);
        bodyPart.turn(TurnDirection.FORWARD, 0.25);
        this.shake(bodyPart);
        this.turnAround();
        this.delay(2.0);
    }

    public void shake(final SJoint bodyPart) {
        bodyPart.turn(TurnDirection.BACKWARD, 0.25);
        bodyPart.turn(TurnDirection.RIGHT, 0.125);
        bodyPart.turn(TurnDirection.LEFT, 0.25);
        bodyPart.turn(TurnDirection.RIGHT, 0.25);
        bodyPart.turn(TurnDirection.LEFT, 0.125);
        bodyPart.turn(TurnDirection.FORWARD, 0.25);
    }

    public void turnAround() {
        this.getRightShoulder().turn(TurnDirection.BACKWARD, 0.25);
        this.getLeftShoulder().turn(TurnDirection.BACKWARD, 0.25);
        this.turn(TurnDirection.LEFT, 1.0);
        this.getLeftShoulder().turn(TurnDirection.FORWARD, 0.25);
        this.getRightShoulder().turn(TurnDirection.FORWARD, 0.25);
    }

    public void hokeyPokey() {
            this.hokeyPokeyVerse(this.getRightHip());
            this.hokeyPokeyVerse(this.getLeftHip());
            this.hokeyPokeyVerse(this.getRightShoulder());
            this.hokeyPokeyVerse(this.getLeftShoulder());
    }
```

Run your animation to ensure that it still works. Close your NetBeans project.

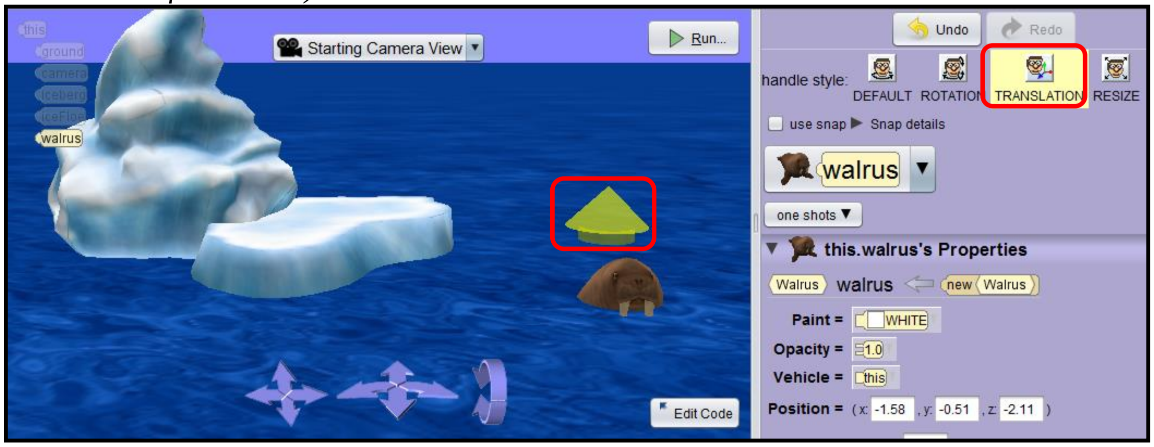
Using Alice Built-in Functional Methods

Our goal is to have a walrus hop onto an ice floe by using built in Alice functional methods. You can use the built in functions answer questions you have about your objects. For example, the **getDistanceTo** function will return the distance from one object to another.

1. Get into **Alice**.

2. Choose the **Sea_Surface** template.

3. Click on the Setup Scene button and add a walrus, an iceberg, and an ice floe to your world. You can leave the default names for the objects, or create your own names. Just be sure to follow the naming rules if you are going to name them yourself. Place the walrus into the water with his head sticking out as shown below *(you will need to click on the Translation button and use the arrow above the walrus to move the walrus so that just his head is out of the water).*



Walrus walrus ⇦ new Walrus

Paint = WHITE

Opacity = 1.0

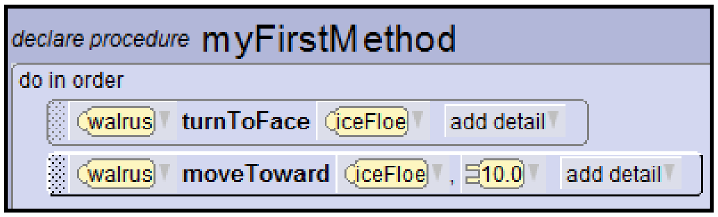
Vehicle = this

Position = (x: -1.58 , y: -0.51 , z: -2.11)

4. Save your project as **WalrusToIceFloe**. Go back to the code editor.

5. Select the walrus and then drag the **turnToFace** method onto the editor and select the **iceFloe** object as the argument.

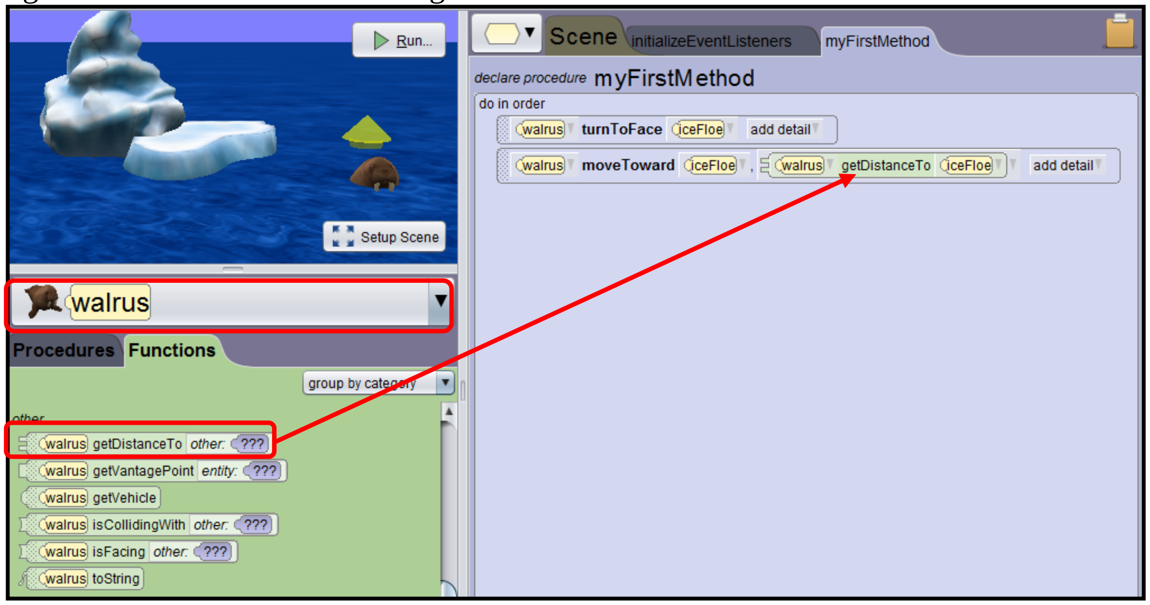
6. Next, drag the **moveToward** method onto the editor selecting the **iceFloe** as the first argument and **10** meters as the second argument as shown below.



```
declare procedure myFirstMethod
do in order
    walrus turnToFace iceFloe add detail
    walrus moveToward iceFloe , 10.0 add detail
```

7. Click the **run** button to watch the animation. The walrus bypasses the iceFloe object and is floating in the air. This tells us that 10 meters is too large of a number, but I still don't know how far the iceFloe truly is. We would need to keep guessing and running the animation to see if we choose the right number. This trial and error process can be very frustrating and is unneccessary. Alice has a built in function called getDistanceTo that will measure the distance from one object to another and return the result.

8. Let's use the **getDistanceTo** function *(function tab)* to find the actual distance between the **walrus** and the **iceFloe** object. You will need to select the walrus, click on the functions tab, and then drag the getDistanceTo function to replace the 10 meters argument. Select iceFloe as the argument.

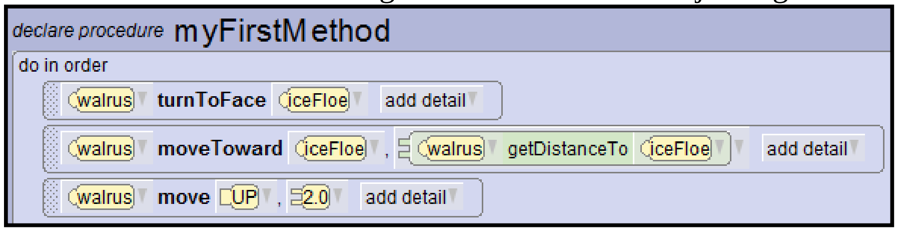


```
declare procedure myFirstMethod
do in order
    walrus turnToFace iceFloe add detail
    walrus moveToward iceFloe , walrus getDistanceTo iceFloe add detail
```

```
walrus getDistanceTo other: ???
walrus getVantagePoint entity: ???
walrus getVehicle
walrus isCollidingWith other: ???
walrus isFacing other: ???
walrus toString
```

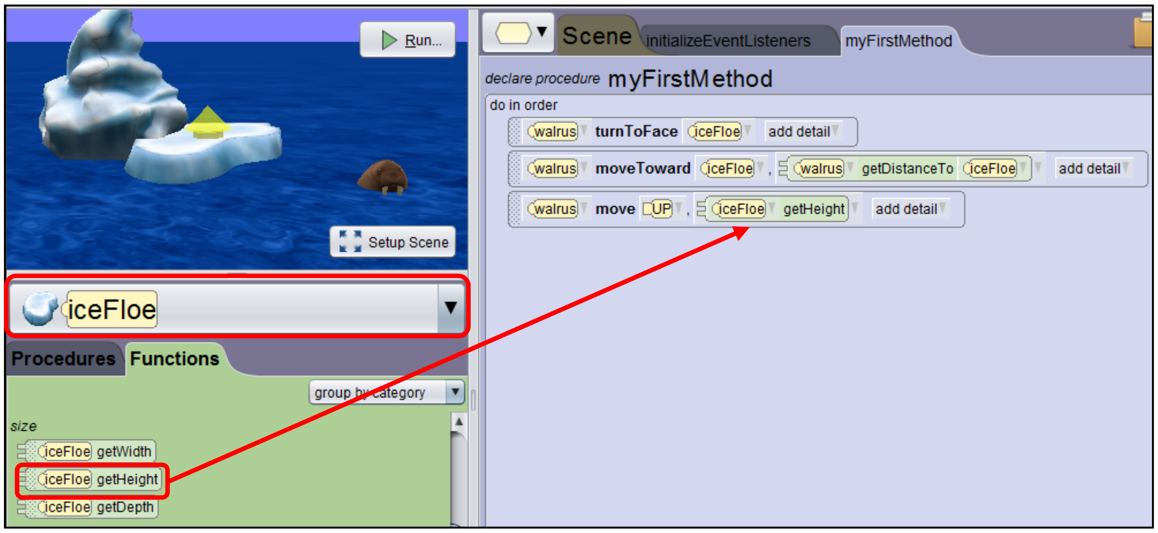
9. Click **run** to view the animation. The walrus ends up inside the iceFloe object. This is okay because we are going to add a move up method that fixes this to look like he is jumping onto the iceFloe object.

10. Go ahead and drag the **move** method onto the editor and select **up** as the first argument and **2** meters as the second argument. The 2 meters is just a guess.



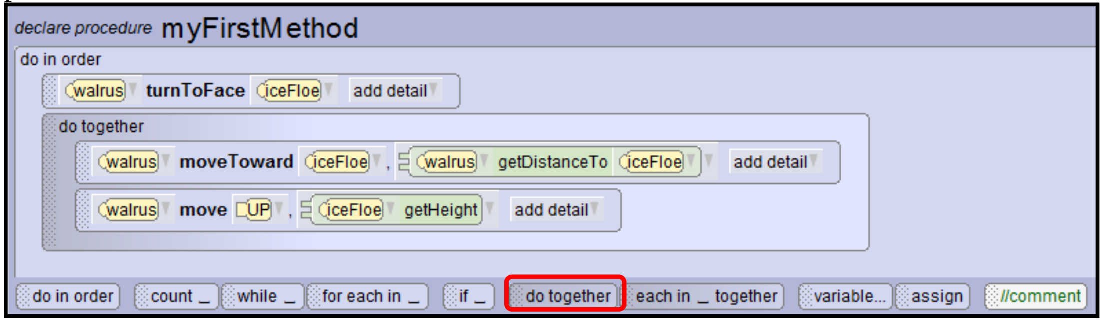
declare procedure myFirstMethod
do in order
walrus turnToFace iceFloe add detail
walrus moveToward iceFloe , walrus getDistanceTo iceFloe add detail
walrus move UP , 2.0 add detail

11. Click **run**. The 2 meters is too far. Again, we don't want to spend time playing this guessing game of how far objects are and how tall they are; there are built in Alice functions that will give you this information so that we don't have to guess. Click on the iceFloe object, then click on the functions tab, and then drag the **getHeight** function *(function tab)* for the iceFloe to replace the 2 meters for the distance on the move method.



declare procedure myFirstMethod
do in order
walrus turnToFace iceFloe add detail
walrus moveToward iceFloe , walrus getDistanceTo iceFloe add detail
walrus move UP , iceFloe getHeight add detail

12. Click **run** to view the animation. The walrus ends up on the ice floe, but he goes right through it and this doesn't look very natural. If we change the moveToward and move method so that they happen at the same time, this will fix it so that he moves at a diagonal and doesn't end up inside the ice floe. Drag the **do together** block onto the editor and place the **moveToward** and **move** methods inside of this block.



declare procedure myFirstMethod
do in order
walrus turnToFace iceFloe add detail
do together
walrus moveToward iceFloe , walrus getDistanceTo iceFloe add detail
walrus move UP , iceFloe getHeight add detail

13. Click **run** to view the animation. The walrus should now jump onto the ice floe and not through it. Save and close this project.

Determining the Tallest Object

Our goal is to use conditionals to determine the tallest object.

1. Open Alice3.

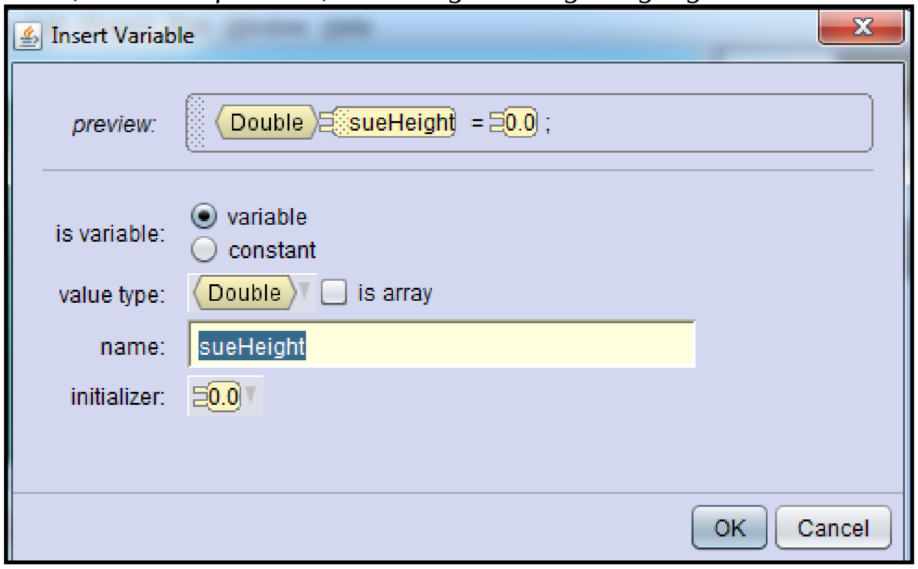
2. Scene setup: Begin with a grass template. Add 3 adult female objects from the Biped classes (change the characteristics of each person to make them unique). Give them the names of sue, betty, and kelly respectively. Move them away from each other.

3. Name this project **TallestPerson**.

4. Switch your preferences to **Java view** instead of Alice view. Click on Window from the menu, then Preferences, Programming Language, and select Java.

5. Declare a variable to represent sue's height. Call it **sueHeight**, make it a Double and set the initializer to 0.0 as follow:

   Note: If your version is saying DecimalNumber instead of Double, click on Window menu, select Preferences, then Programming Language and then JAVA.


Insert Variable

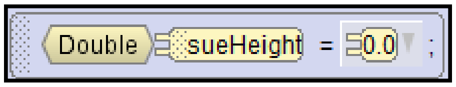
preview: Double sueHeight = 0.0 ;

is variable: (●) variable ( ) constant

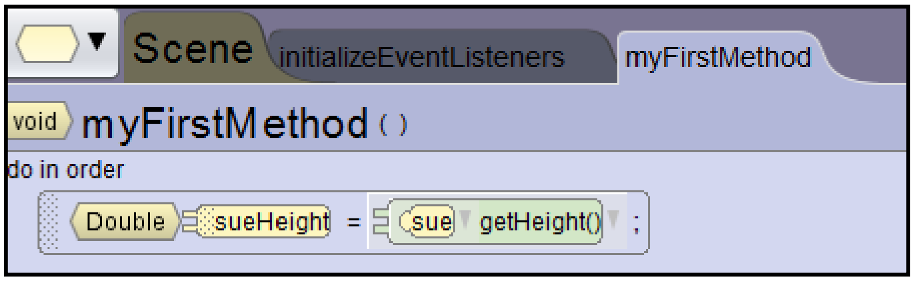
value type: Double ☐ is array

name: sueHeight

initializer: 0.0

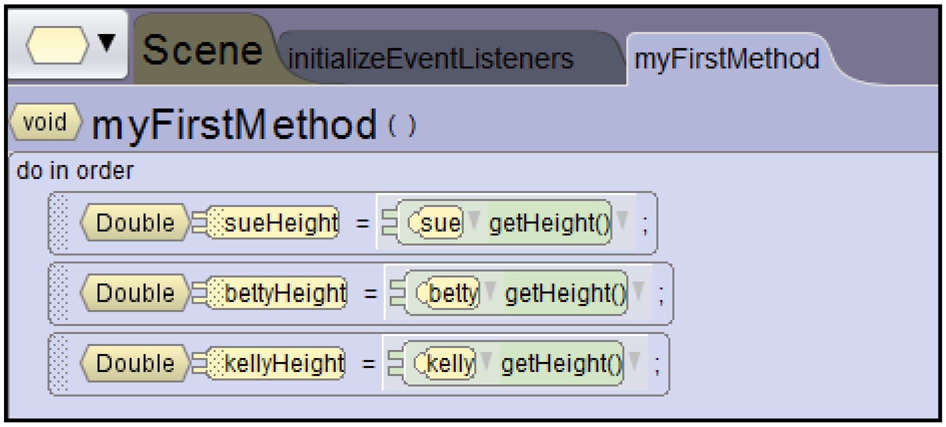

Double sueHeight = 0.0 ;

6. Click on the **Functions** tab for **sue** and drag **sue.getHeight** to replace **0.0** in your variable statement for sueHeight as shown below:



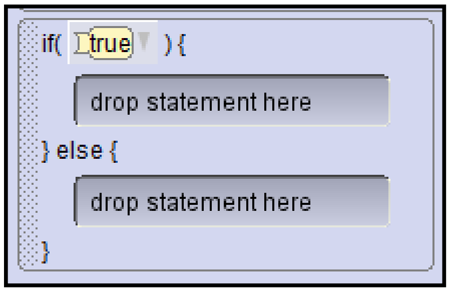
```
void myFirstMethod ( )
do in order
    Double sueHeight = sue getHeight() ;
```

7. Repeat this for betty and kelly. *(Make sure you are clicked on the correct girl when doing her height).* You should have 3 variables (sueHeight, bettyHeight, and kellyHeight) with each of the object's height saved accordingly.



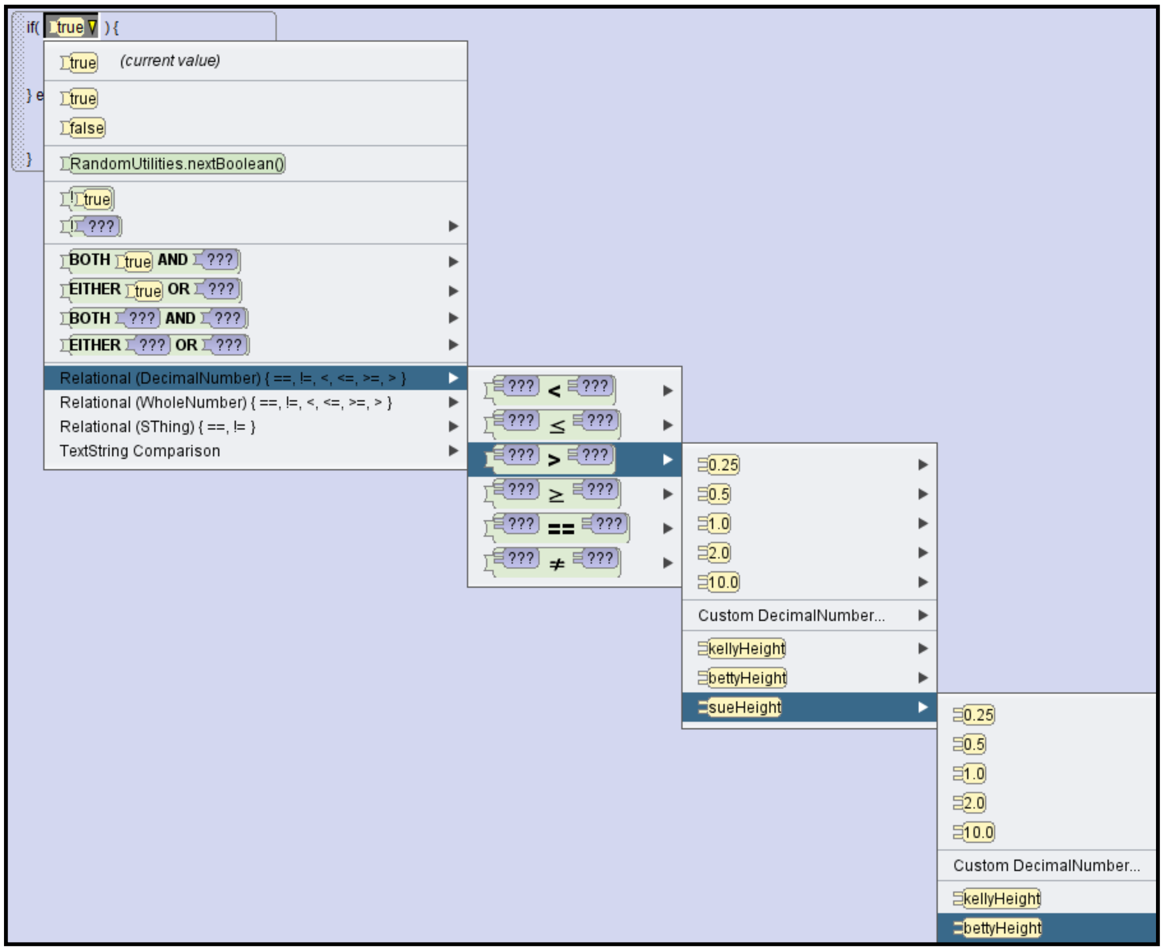
```
void myFirstMethod ( )
do in order
    Double sueHeight = sue getHeight() ;
    Double bettyHeight = betty getHeight() ;
    Double kellyHeight = kelly getHeight() ;
```

8. Now we need to figure out which of the girls is the tallest. This would be easy if we only had 2 girls, but since we have 3 it is going to be a bit more complex. Let's start with sue and see if she is the tallest. We need to figure out if sue is taller than betty first. Drag up the **if… block** statement and release and set to true as follows:

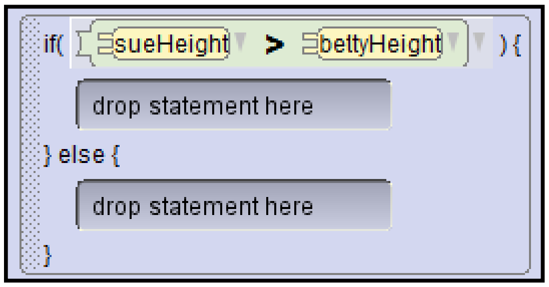


```
if( true ){
    drop statement here
} else {
    drop statement here
}
```

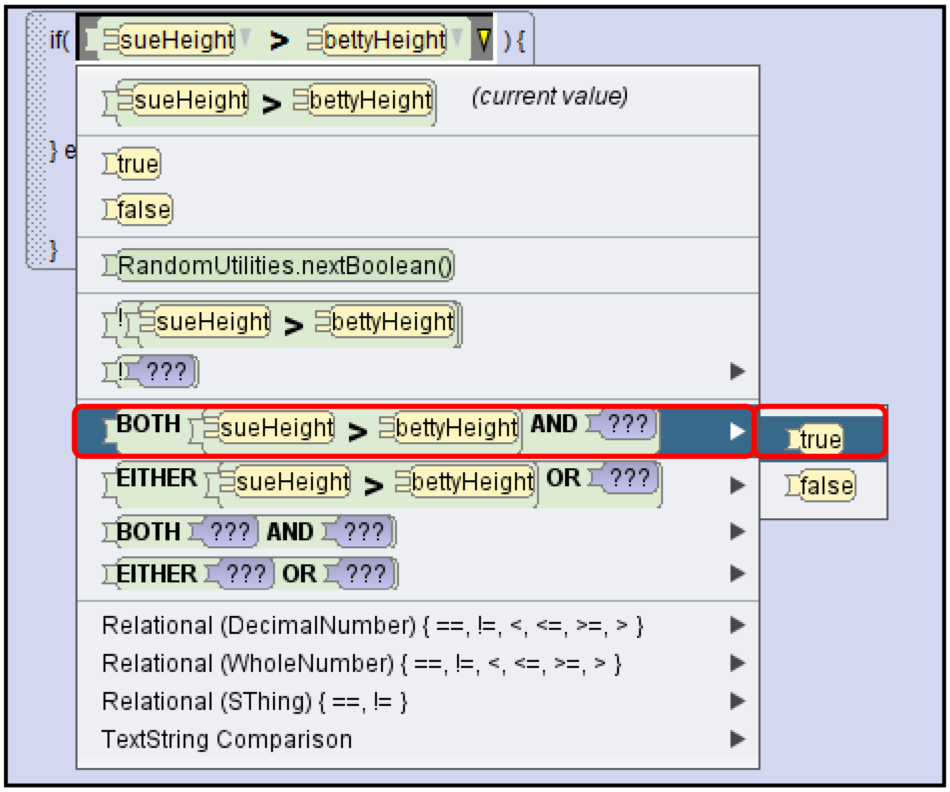
9. You should change the **true** by choosing **Relational Decimal Number**, then **???> ???** then **sueHeight** and then **bettyHeight** as follows



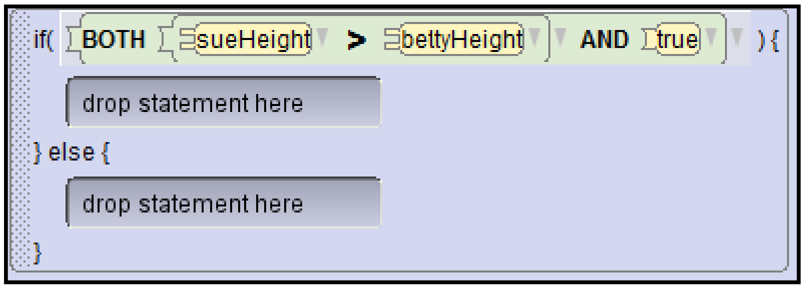
10. So that your if statement should look as follows:


```
if( sueHeight > bettyHeight ){
    drop statement here
} else {
    drop statement here
}
```

11. Just because sue is taller than betty doesn't necessarily mean that she is taller than kelly. We need to add another condition (compound condition) to ensure that sue is taller than both of betty and kelly. You need to have both conditions be true so you will click on the down arrow farthest to the right, select **BOTH sueHeight>bettyHeight AND ???** then select **true**.

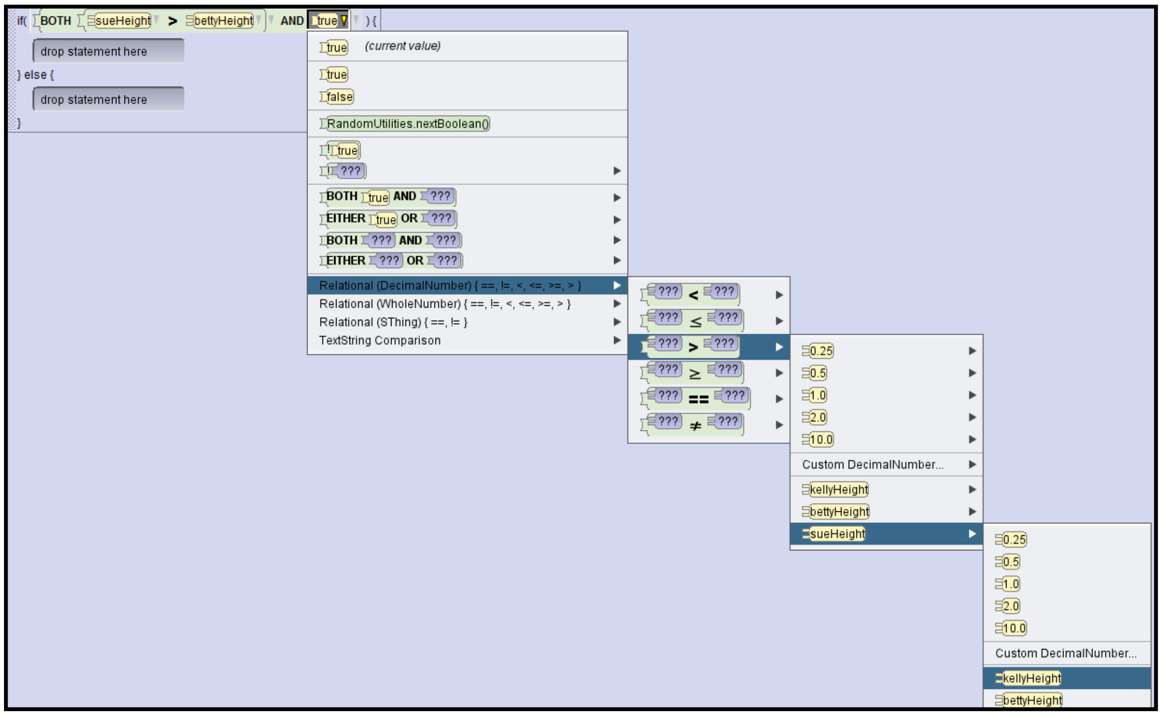

if(sueHeight > bettyHeight){
sueHeight > bettyHeight (current value)
true
false
RandomUtilities.nextBoolean()
! sueHeight > bettyHeight
! ???
BOTH sueHeight > bettyHeight AND ???
true
false
EITHER sueHeight > bettyHeight OR ???
BOTH ??? AND ???
EITHER ??? OR ???
Relational (DecimalNumber) { ==, !=, <, <=, >=, > }
Relational (WholeNumber) { ==, !=, <, <=, >=, > }
Relational (SThing) { ==, != }
TextString Comparison

You should have the following code:



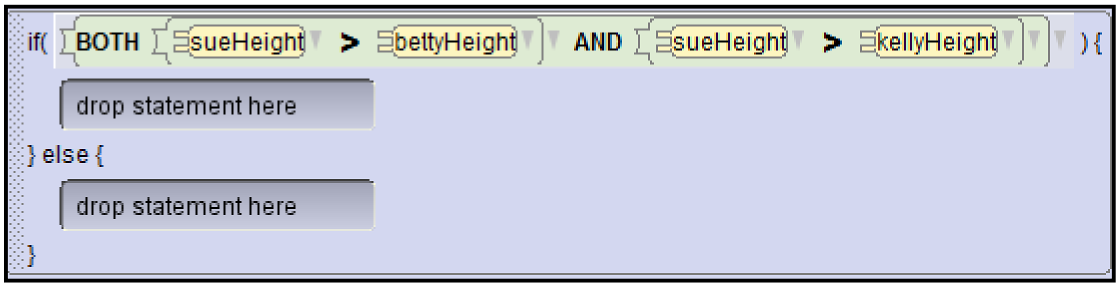
```
if( BOTH sueHeight > bettyHeight AND true ){
    drop statement here
} else {
    drop statement here
}
```

12. Click on the down arrow next to the “true” that is next to the “AND”. Then select **Relational DecimalNumber**, then **???> ???**, then **sueHeight**, and then **kellyHeight**. Your if statement should now look as follows:



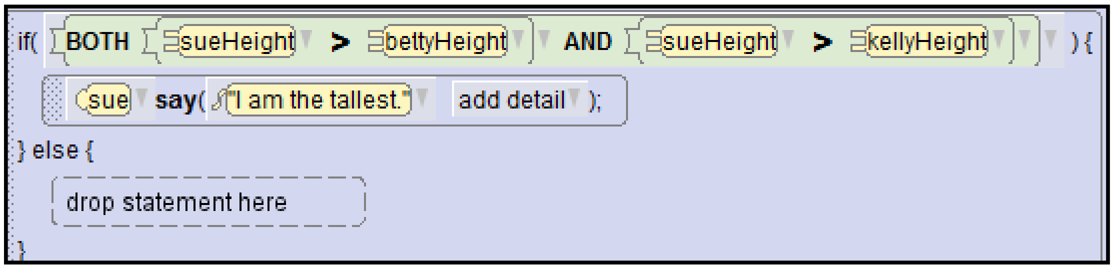
```
if( BOTH sueHeight > bettyHeight AND true ){
    drop statement here
} else {
    drop statement here
}
```

13. You should now have the following if statement:



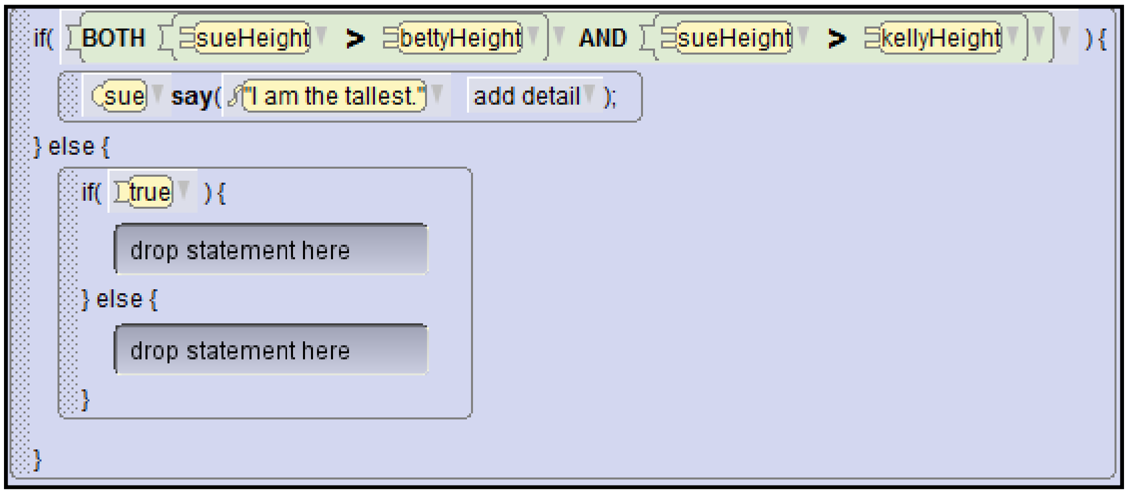
```
if( BOTH sueHeight > bettyHeight AND sueHeight > kellyHeight ){
    drop statement here
} else {
    drop statement here
}
```

14. We can now say that sue is the tallest if she is taller than betty and taller than kelly. Have sue say that she is the tallest as shown below*:*



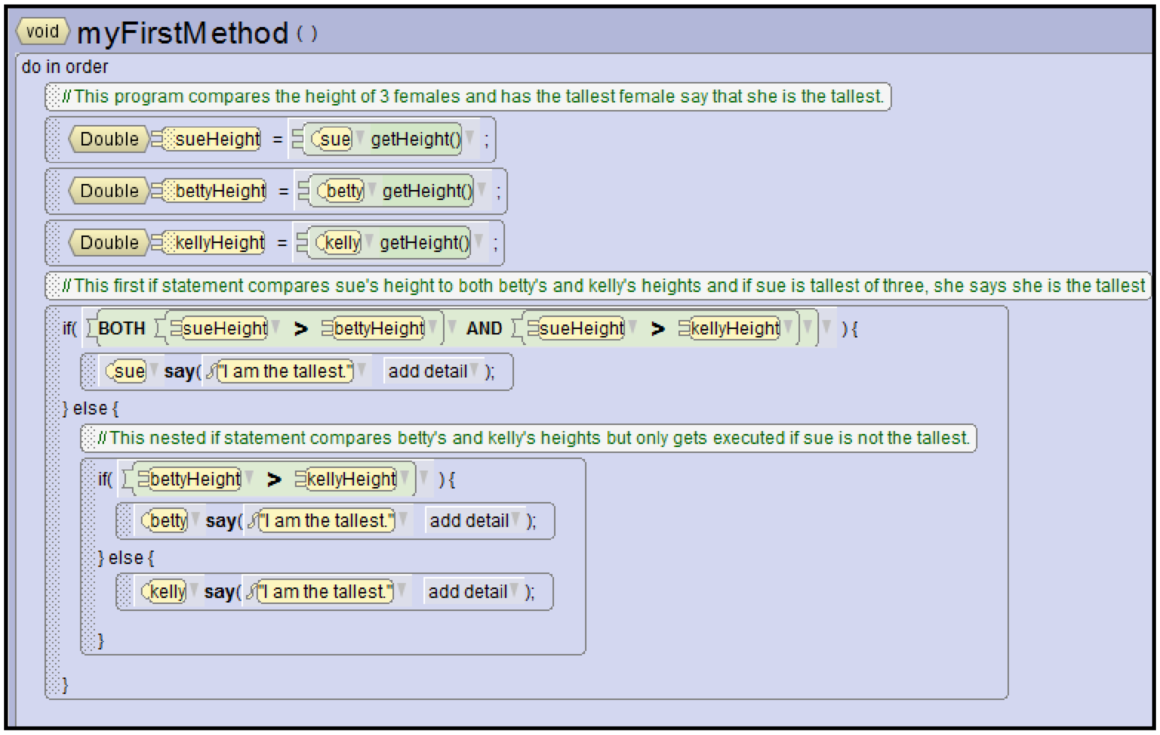
```
if( BOTH sueHeight > bettyHeight AND sueHeight > kellyHeight ){
    sue say( "I am the tallest." add detail );
} else {
    drop statement here
}
```

15. In the “else” block of this if statement, we know that sue must not be the tallest or we would have never gotten to the else block. However, we don’t know which of the other two, betty or kelly, is the tallest. We need a nested if statement to determine that. We only need to compare betty and kelly. Drag an **if** tag into the else block and set it to **true** as follows:



```
if( BOTH sueHeight > bettyHeight AND sueHeight > kellyHeight ){
    sue say( "I am the tallest." add detail );
} else {
    if( true ){
        drop statement here
    } else {
        drop statement here
    }
}
```

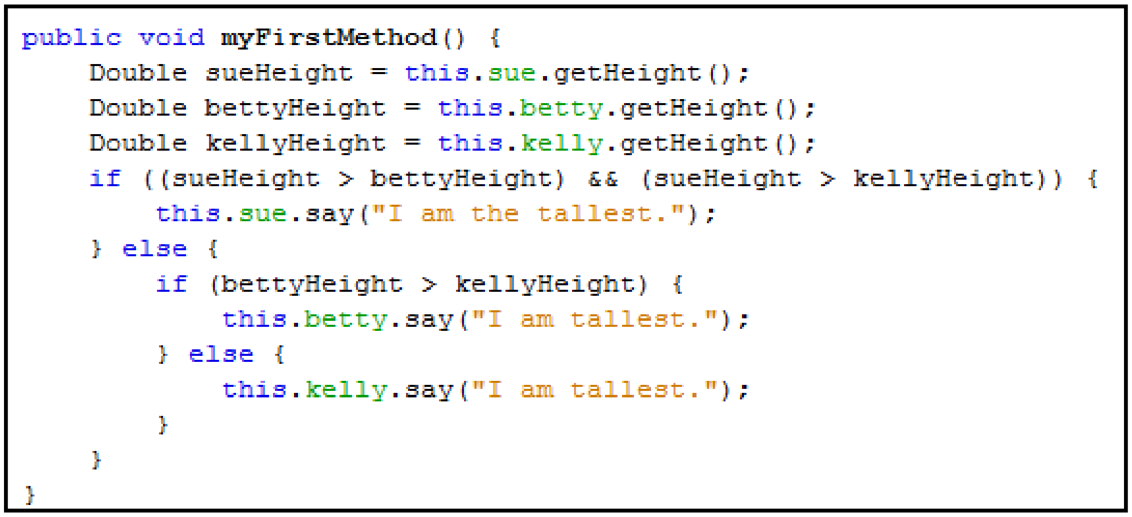
16. Now, on your own, have the computer compare betty and kelly’s height. If betty’s height is greater than kelly’s height, then betty should say she is tallest, else kelly should say she is tallest. You should also add some comments and your final version of the program should look as follows:



```
void myFirstMethod ( )
do in order
    // This program compares the height of 3 females and has the tallest female say that she is the tallest.
    Double sueHeight = sue getHeight() ;
    Double bettyHeight = betty getHeight() ;
    Double kellyHeight = kelly getHeight() ;
    // This first if statement compares sue's height to both betty's and kelly's heights and if sue is tallest of three, she says she is the tallest
    if( BOTH sueHeight > bettyHeight AND sueHeight > kellyHeight ){
        sue say( "I am the tallest." add detail );
    } else {
        // This nested if statement compares betty's and kelly's heights but only gets executed if sue is not the tallest.
        if( bettyHeight > kellyHeight ){
            betty say( "I am the tallest." add detail );
        } else {
            kelly say( "I am the tallest." add detail );
        }
    }
```

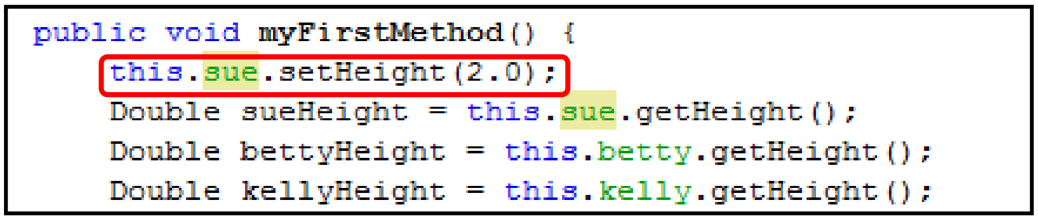
17. Run your program. Does it tell you who is the tallest? Try resizing each of the girls and see if the results change. Make sure that you try all possibilities to truly test your program. Save your program. *Note: you may have to resize the animation window to see the speech bubbles and/or change the duration of the say method to have enough time to read the text in the speech bubbles.*

18. Get into NetBeans and start a new project based on the **TallestPerson.a3p** file and then you should scroll down in the **Scene.java** file until you can see the **myFirstMethod** method as follows:



```
public void myFirstMethod() {
    Double sueHeight = this.sue.getHeight();
    Double bettyHeight = this.betty.getHeight();
    Double kellyHeight = this.kelly.getHeight();
    if ((sueHeight > bettyHeight) && (sueHeight > kellyHeight)) {
        this.sue.say("I am the tallest.");
    } else {
        if (bettyHeight > kellyHeight) {
            this.betty.say("I am tallest.");
        } else {
            this.kelly.say("I am tallest.");
        }
    }
}
```

19. To test the program, let's make sue be the tallest by inserting a statement at the beginning of the myFirstMethod that will adjust sue's height to be 2.0 meters as follows:



```
public void myFirstMethod() {
    this.sue.setHeight(2.0);
    Double sueHeight = this.sue.getHeight();
    Double bettyHeight = this.betty.getHeight();
    Double kellyHeight = this.kelly.getHeight();
```

20. Run your program. Unless you have made your females very tall, this should be enough to make sue the tallest. If not, make the number larger than 2.0. Once it works for sue being the tallest, adjust that first line to set the height for betty to be a large number and see if it works for her. Last of all, do a test by adjusting kelly's height and see if it works for her.

21. Close your project.

Exploring Alice on your own

O\Now that you have learned how to do many interesting things in Alice 3.0 it is time to work on your own creations!

First you should create a story on paper, or at least an outline. Then you can start selecting characters and building scenes. Then make your characters move and talk, and create your transitions. Don't forget the title screen and the credits at the end!

At the end of this sections we will watch the animations that you have made.

Acknowledgements

Thanks to Dr. Tebring Daly at Collin College for the use of this material. These materials were developed through a National Science Foundation - Advanced Technological Education (ATE) Grant #0835036.