# Teaching With Alice
# First Bytes
# Teachers Workshop

July 2008



## Topics

- What is Alice?
- What resources are available?
- How is Alice used in teaching?
- Demo of Alice programming



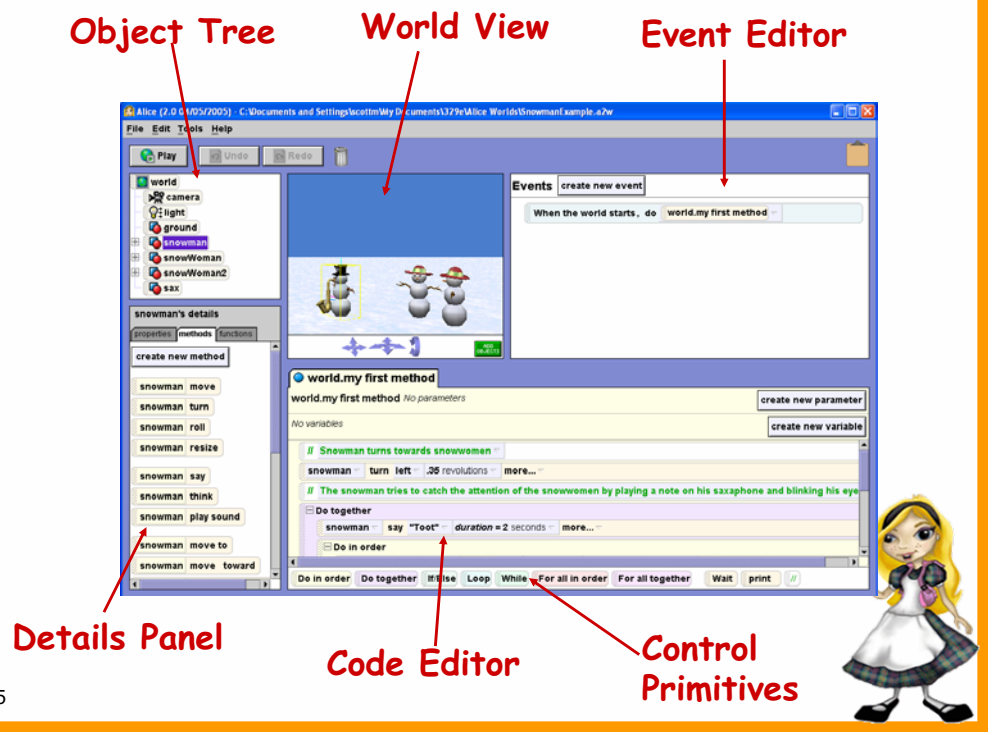
## What is Alice?

- Alice is a visual programming language.
- Alice is an object based language. The objects in Alice are 3 dimensional models.
- The output of Alice programs are 3 dimensional movies.



## Visual Programming

- Programming is done by pointing and clicking, dragging and dropping, selecting from menus, and some typing
- Syntax errors removed from the equation
  - no braces, no semi colons





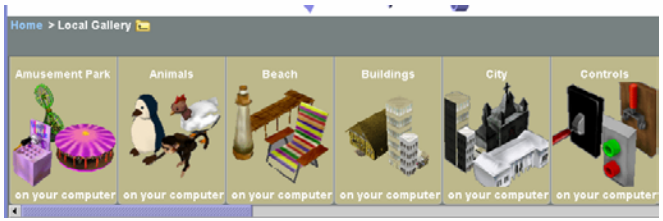

# Object Based Programming

- Built in library of models.
- More available on the web.
- All objects have certain methods and behaviors
  - move, turn, say, roll, resize
- New methods can be added to an object
  - object can be saved as a new class
- Polymorphism is not supported.



# Alice Models

- Main programming data are 3d models
- Many built in and more on web







# Output

- Output are 3d movies
  - run the program, play a movie
  - can also add sound to programs
- A lot easier to recognize logic errors
  - "Why do my ninja's arms keep flying away?"

# Alice Resources

- Main page
  - www.alice.org
- download Alice 2.0 for free
- story telling Alice for middle school
- Models gallery
- Forums
- Textbooks list



# Instructional Materials

- www.aliceprogramming.net
- Password protected
  - userid:
  - password:
- Workshop schedule
- Example course calendars / syllabi
- Slides and sample worlds
- Solutions to chapter exercises and projects (Dann, Cooper, Pausch book)
- Sample exams and test bank questions



# Even More Materials

- Dick Baldwin, ACC teacher
  - www.dickbaldwin.com
  - www.dickbaldwin.com/tocalice.htm
- Lots of materials and "how to's"
- Alice newsletter. To sign up contact Barbara Conover
  - bconover@sju.edu



# How is Alice Used in Teaching

- Originally designed for students in middle school
- Has been successful with older students
- Used in lots of types of courses
  - computer literacy
  - pre cs or pre AP
  - cs1 or APCS
  - programming for non CS majors

# Approaches

- Cover basics, chapters 1 and 2 quickly
  - learning the tool
- Paths through intro programming
  - objects early (control structures first)
  - objects first
  - objects first, recursion early
- Interactivity
  - can create animations / movies only
  - OR introduce events and interactivity



# Projects

- Closed-ended
  - write a program to meet specified criteria
  - allows focusing on some aspect of programming
  - closed-ended with options - charades
- Open-ended
  - some students show great creativity here
  - some make very skimpy programs
  - chance to require storyboarding and planning



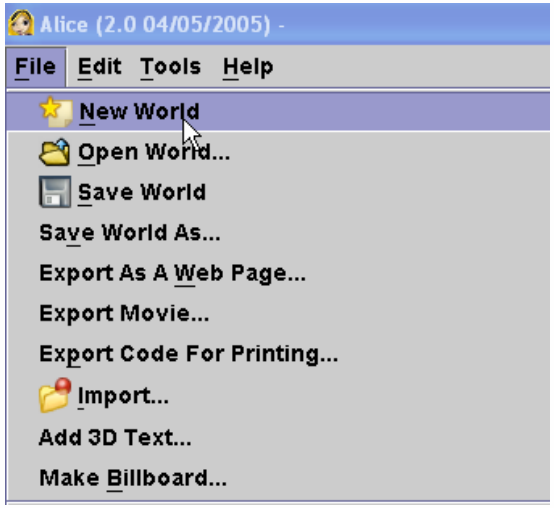
# Sample Program - Bunny and Broccoli



# Demo of Alice Programming

- Follow along!
- Problem solving and programming in Alice
  - given a scenario create program to enact the story
- A bunny is sitting in a field. Around the bunny broccoli sprouts and grows. The bunny hops over to the closest broccoli plant and eats it.

# Create a New World

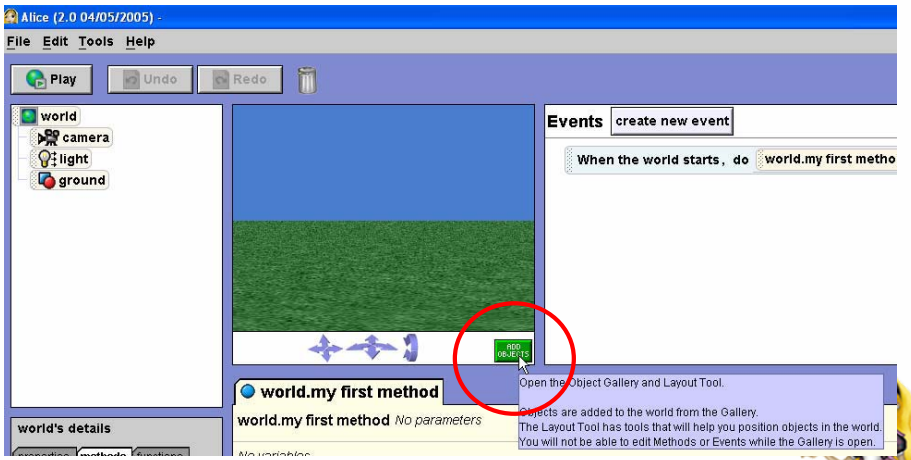





# Select Template (Ground)

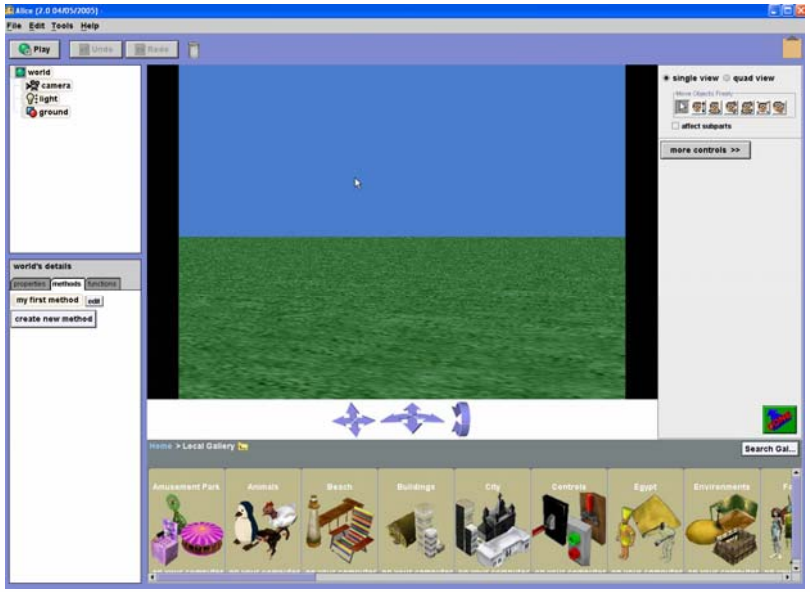



# Add Objects






# The Scene Editor

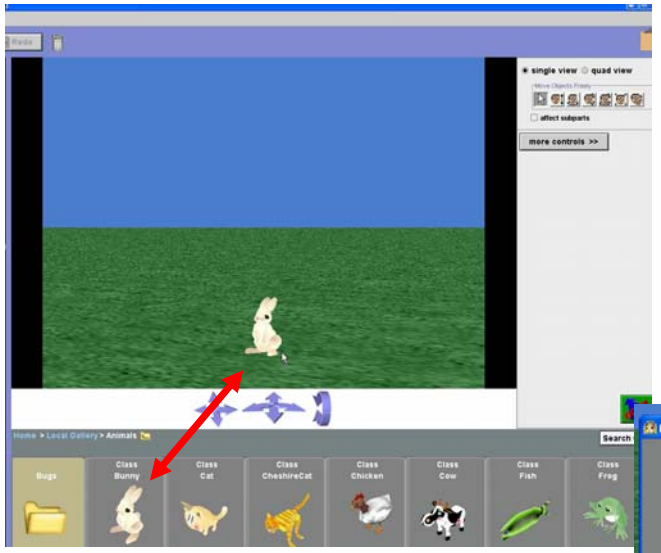


# Beware the Scene Editor

- Students can spend A LOT of time in the scene editor setting up and tweaking a world
- Is that really programming?
  Or computer science?
  Or Computational thinking?



# Add Objects




- Drag and Drop
- Click on picture then click on **Add Instance**



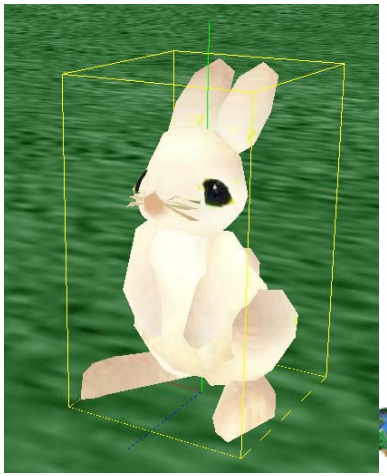
# Objects in The World

- Objects in Alice
  - Have their own frame of reference
  - forward – backwards
  - up – down
  - left – right

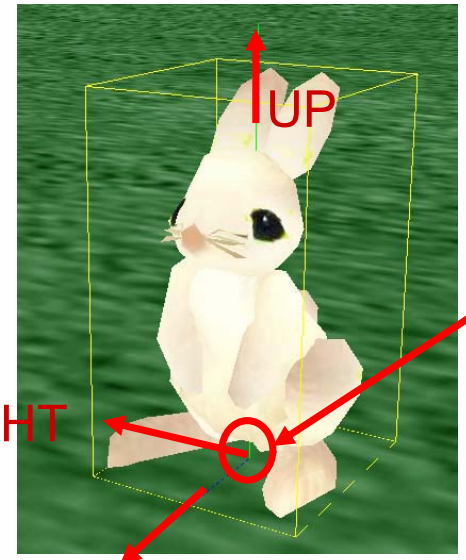


# Frame of Reference

- Clicking on object bring sup its *bounding box*
- Can also see center point
- .. and axes

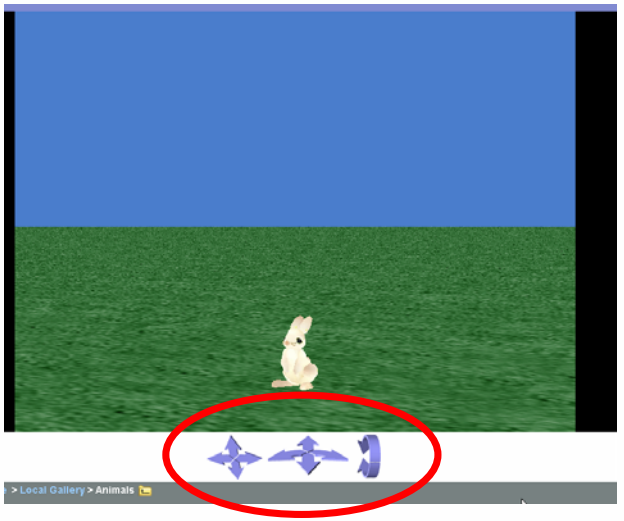


# Frame of Reference






# Camera Controls



Alter position of camera with these controls.



# Mouse Control Tools Kit


Default. Move selected object left, right, forward, backwards.

Copy objects.

Move selected object up and down.

Resize objects.

Tumble objects.

Turn object left and right.

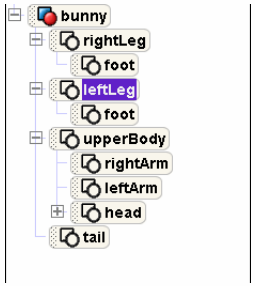
Turn object forwards and backwards.

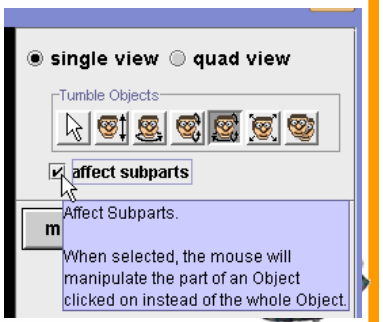
CTRL Z or Undo Button to undo mistakes!



# Subparts

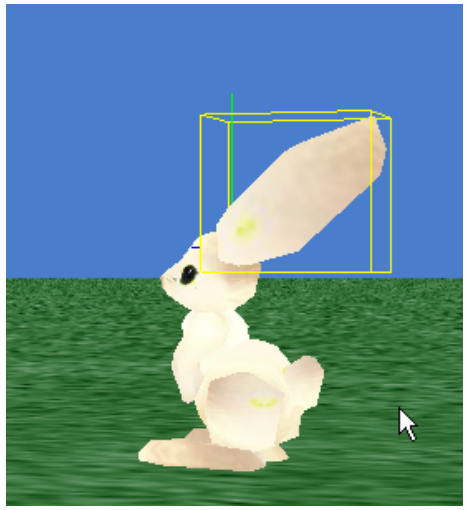
- Objects often have sub parts
  - may have their own frame of reference
- Clicking **affect subparts** box allows selection and movement of subparts



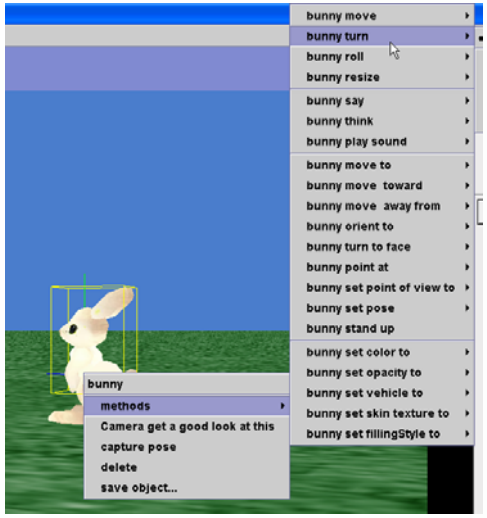
# Subparts

- Bigger ear





# Alternate Positioning Techniques

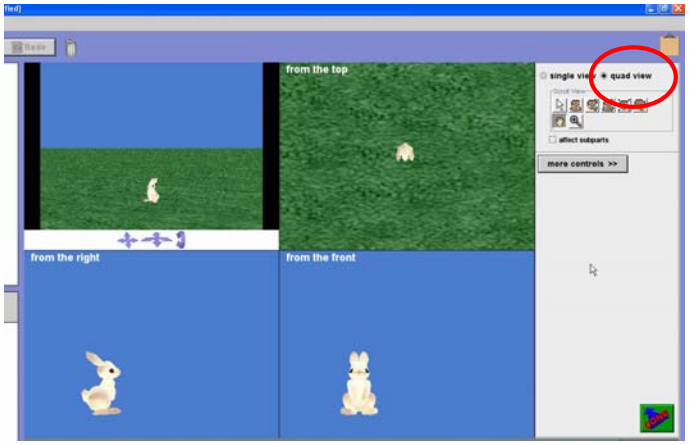



- Right click on object in world on object tree and select method
- Drag and drop method from the details panel.



# Quad View

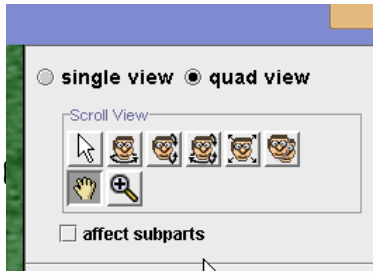
- Use world's absolute frame of reference to view relative position of objects






# Finding Objects

- To reposition in a quad view
  - select zoom in and out fro mouse controls
  - zoom way out
  - select scroll from mouse controls to center objects
  - zoom back in

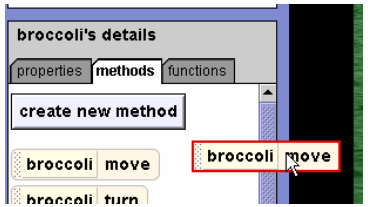



# Setting Up Initial Scene

- Add bunny
- Add broccoli
  - local gallery -> kitchen -> food
- Make broccoli bigger
- Move broccoli below the ground
  - How to simulate "growing"?
  - move down exactly 1/2 meter using drop down menus or drag and drop

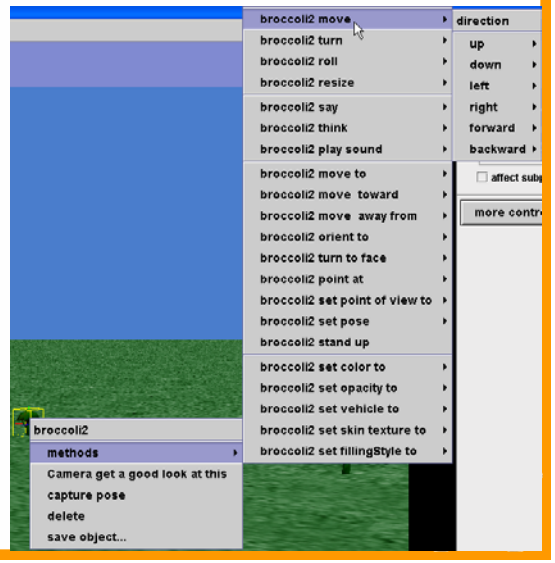


# Moving Broccoli Down




Option 1

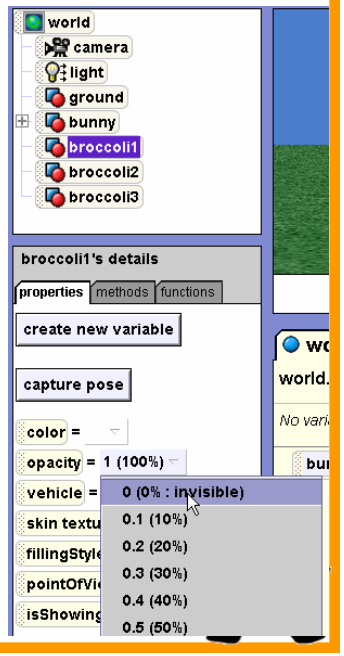
Option 2






# Making Broccoli Invisible

- In our program we want the broccoli to grow.
- We will do this by having it
  - move up
  - get bigger
  - become visible
- Need to make the broccoli invisible
- Select each broccoli from the object tree and click the properties tab
- Change ***opacity*** from 100% to 0%






# Back to Programming View

- When setup complete click the green done button to go back to the programming view.


# Programming the World

From a storyboard to a program.



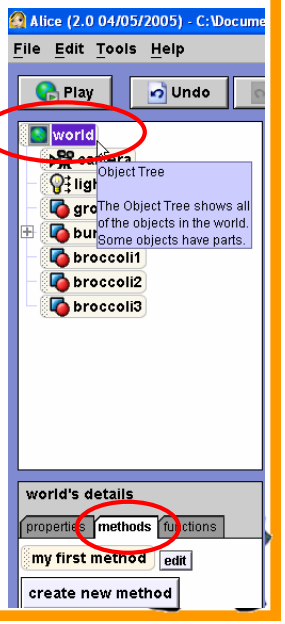
# Recall the Storyboard

- A bunny is sitting in a field.
  Around the bunny broccoli sprouts and grows.
  The bunny hops over to the closest broccoli plant and eats it.
- Let's add some detail at the start of the movie.
  - The bunny first turns to fast the camera. Then the broccoli start to grow and while it grows the bunny hops up and down.



# Methods

- Select the world object from the object tree and the methods tab in the details panel.
- The world starts with a single method, "my first method"
- Like main in a Java or C++ program.






# Adding Commands to Methods

- If the "my first method" is not displayed in the code editor click the edit button next to the method in the detail panel.
- Commands are added by dragging and dropping them into a method.
- Select the bunny from the object tree.
- Drag the ***turn to face*** command into the code editor.

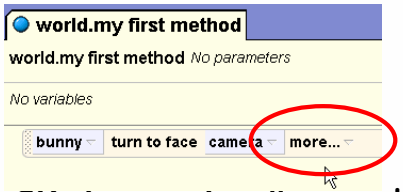
# Adding Commands

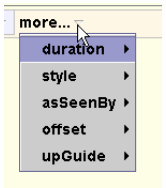

- ***turn to face*** is a method
- When adding a method to the code editor if any parameters are required a menu pops up to select the arguments.
- Select the camera.




# More Parameters

- After adding the bunny.turn to face command the "my first method" will look like this:



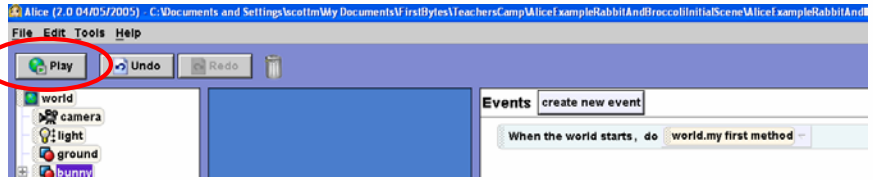




- Click on the "more" option to see what other parameters can be changed
  - duration, style, asSeenBy
  - change duration to 3 seconds



# Test

- Click the play button to see the movie / output of the program.




- "my first method" will execute because of the only event in the program at this point.

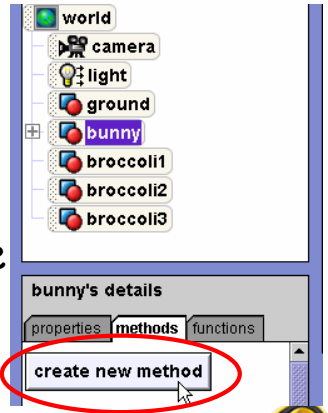


# Adding Behaviors

- Next we want the bunny to hop while the broccoli grows.
- Methods can be ***world level*** or ***class level***.
  - world level methods belong to the world.
    - a method should be world level method if it involves two or more objects
  - class level methods belong to a particular class / object.
    - a method should be a class level method if it involves only one object

# Creating a Hop Method

- The bunny does not have a ***hop*** method so we will create one.
- Select the bunny from the object tree and click on the create new method button in the details panel.





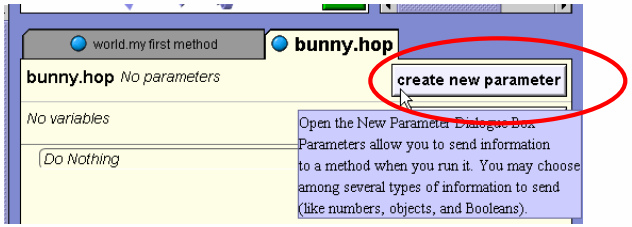

# Creating a Hop Method

- A window pops up asking for the name of the method
  - try various names to see what is a legal identifier and what is not
- After giving the new method a name a new tab pops up in the code editor
- Should hop be one hop or parameterized?
- Should parameter be time to hop or number of hops to make?
- Any other way to make it more general?




# Adding Parameters

- Let's add parameters for distance to hop up and the time to do the hop

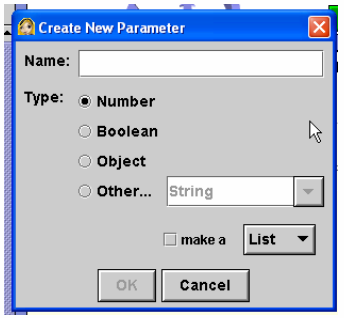



- Click the ***create new parameter*** button in the code editor.



# Adding Parameters

- Give the parameter a name and pick the data type
  - distance -> a Number
  - time -> a Number

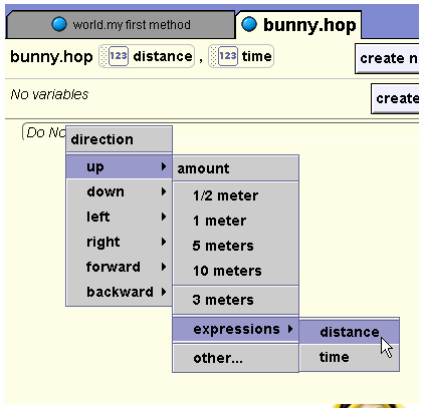




- When called the hop method now requires two parameters

# Adding Commands to Hop

- To hop the bunny will move up and then down.
- Drag the ***move*** command into hop and fill in the parameters.
- Drag another ***move*** command into hop and fill in the parameters.





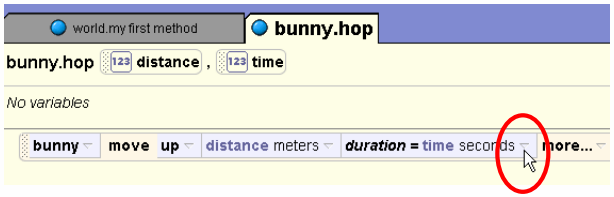

# Adding Commands to Hop

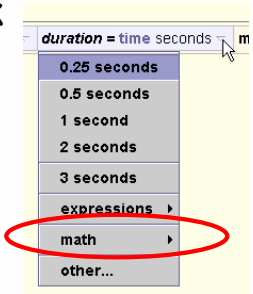
- To change the duration of moving up select the *more* option from the ***move*** command.
- Select ***duration*** then ***expressions*** then ***time*** (or the name of your parameter for time)



# Adding Commands to Hop

- To change the duration of the move to half of the time parameter click on the triangle to open the drop down menu.

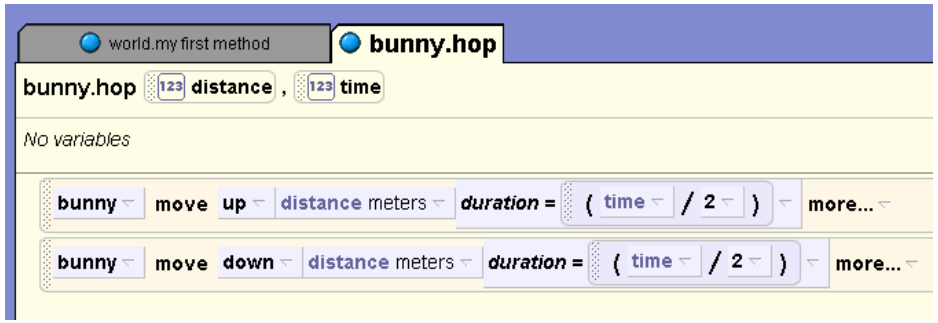



- Select math and divide time by 2.
- Do the same for the move down.



# Completed Hop Method




# Back to my first method

- We want the bunny to hop while the broccoli grows
  - In the initial set up the broccoli is below the ground and invisible
- The broccoli will grow by
  - moving it above the ground
  - resizing it to double it original size
  - making it visible



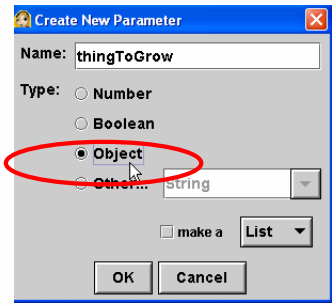
# A grow Method

- Instead of repeating the actions to grow for each broccoli we will put it in a method
  - could make a class level method and then save a new broccoli object that knows how to grow and add two of those to world (inheritance)
  - OR make a world level method and send in each broccoli as a parameter
- We'll take the second option



# A grow Method

- Create a new world level method named grow
- Add a parameter of type Object
- Common mistake is to not change parameter type to correct type.

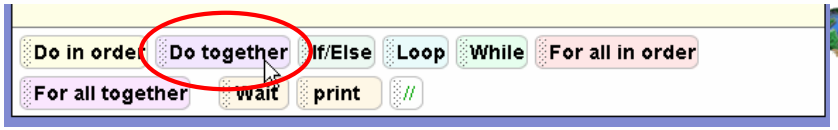






# Adding Commands to Grow

- We want all three things (move up, resize, and become visible) to happen at the same time
- Default for commands is in order
- ***Do together*** is a primitive that executes commands together
- Drag and drop a Do together into the grow method



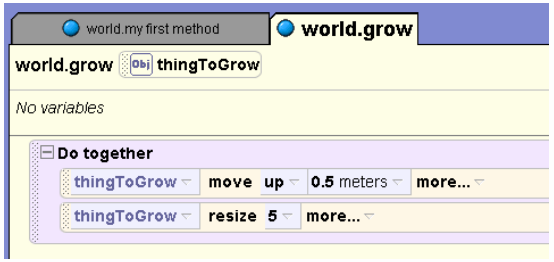

# Do together

- Commands in a Do together block will be executed in parallel
- Each command can have a different duration
- Do together completes when last inner command completes



# Growing

- Drag and drop the parameter from the method header into the Do together block and select the methods to
  - resize
  - move up

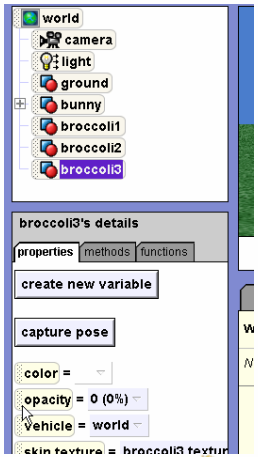



- Change duration to 5 seconds for each



# Becoming Visible

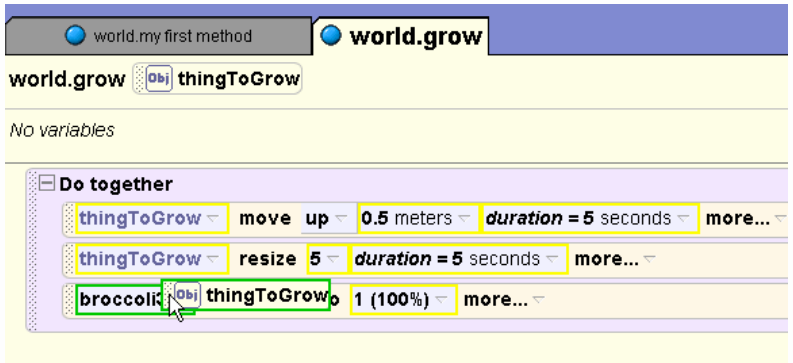
- Properties may be changed as program commands
- A little tricky to do with parameters
- Select any object from the object tree and its properties tab
- Drag the opacity property into the program and select 100%



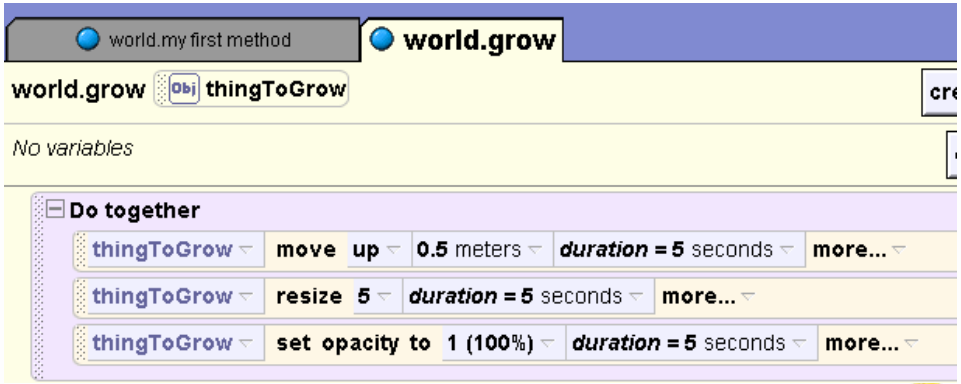



# Becoming Visible

- Now replace the object that we dragged into the grow method with the parameter by dragging and dropping.




# Completed grow Method






# Back to my first method

- Now that the bunny can hop and the broccoli can grow we can complete the first part of the story board
- After the bunny turns to face the camera we want the broccoli to grow and the bunny to hop *all at the same time.*



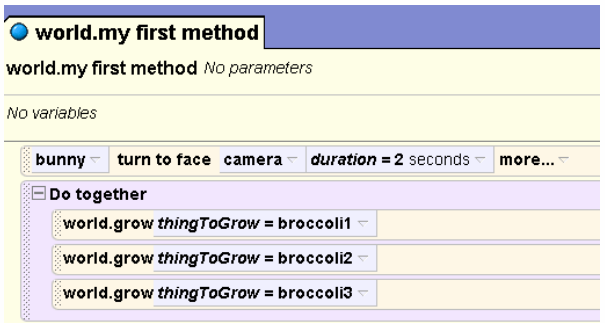

# growing and hopping

- Drag a ***Do together*** block into ***my first method*** after the bunny turns to face the camera
- Drag the ***grow*** method into the ***Do together*** block three times and add pick each broccoli once for a parameter




# Testing

- Test the program by pressing the ***play*** button.




- Is anything wrong?

# Resizing and Moving Up

- Resizing the broccoli has altered the distance of its center point below the ground
- Some of the broccoli's stalk is still below the ground
- Go back to grow method and alter the amount to move up to a value that makes more of the broccoli appear above the ground



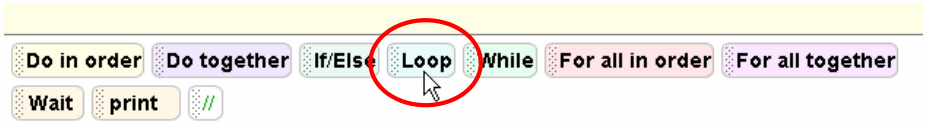
# Hopping

- We want the bunny to hop while the broccoli grows
- Back to my first method
- Broccoli takes 5 seconds to grow
- Have rabbit hop up and down .25 meters at 0.5 seconds per hop
- How many hops?



# Looping

- A counted loop is used when the number of repetitions can be calculated
- Drag a ***Loop*** primitive into the ***Do together*** block

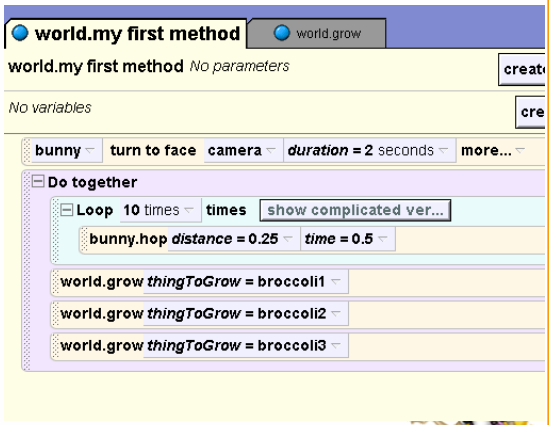



- Number of times to loop is 10



# Hopping

- After Loop is added to Do together drag and drop the bunny hop method into the loop
- Select 0.25 meters for distance to hop and 0.5 seconds for time
- Test!



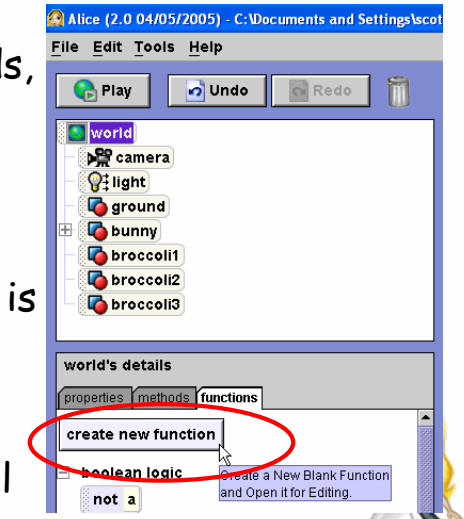

# Eating the Closest Broccoli

- Now we want the rabbit to turn to face the closet broccoli, hop over to it, and eat it.
- Which broccoli is closest?
- We want to be able to reposition broccoli and not have to change program
- Create a ***function*** !



# Creating Functions

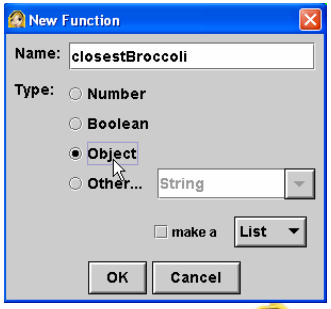
- Functions, unlike methods, return an answer.
- Sometimes called questions.
- Create a function to return the broccoli that is closest to the bunny.
- Select the world in the object tree and the function tab in the detail panel






# Create a New Function

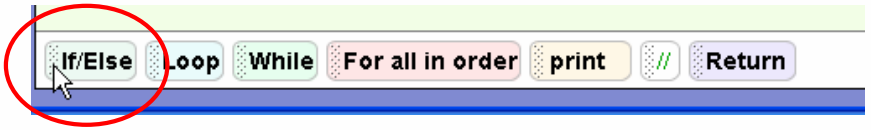
- Click the create ***new function button*** .
- Give the function a name.
- Pick the data type for what the function will return.
- In this case an Object.






# Which Broccoli is Closest

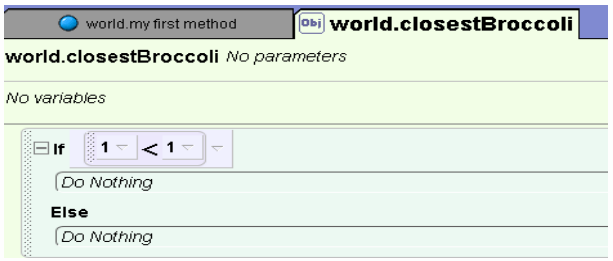
- Decision making for which broccoli is closest
- When is broccoli1 closest?
- Drag an **if/else** into the function




- Initial condition doesn't matter.

# Condition of if/else

- Select world in object tree and function
- replace true in ***if/else*** with ***a<b*** function
- Initial values don't matter

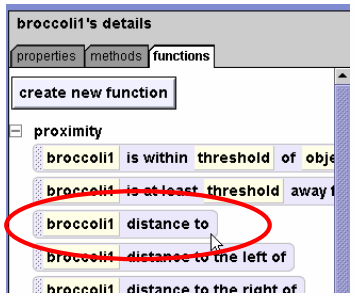






# Checking broccoli1

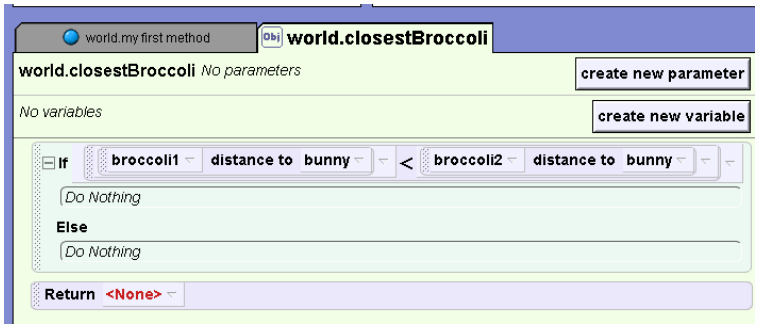
- Click on broccoli1 in the object tree
- Replace the first value in the ***a<b*** with the function ***broccli1 distance to***
- Select the bunny as the parameter






# Checking broccoli1

- Replace the second value of a<b with the distance from broccoli2 to the bunny



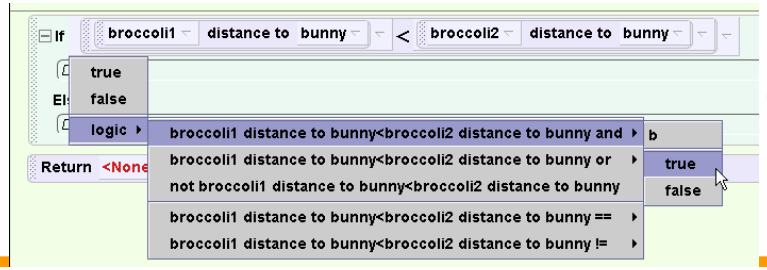

- Multiple ways to go from here

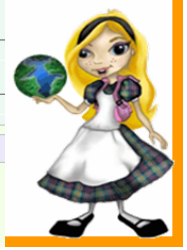


# Checking broccoli1

- One option, AND

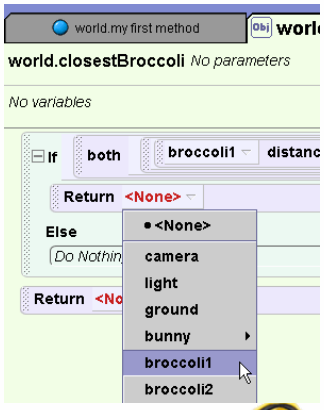

- Bring up drop down menu on expression, select ***logic*** and then the ***and*** option



# Checking broccoli1

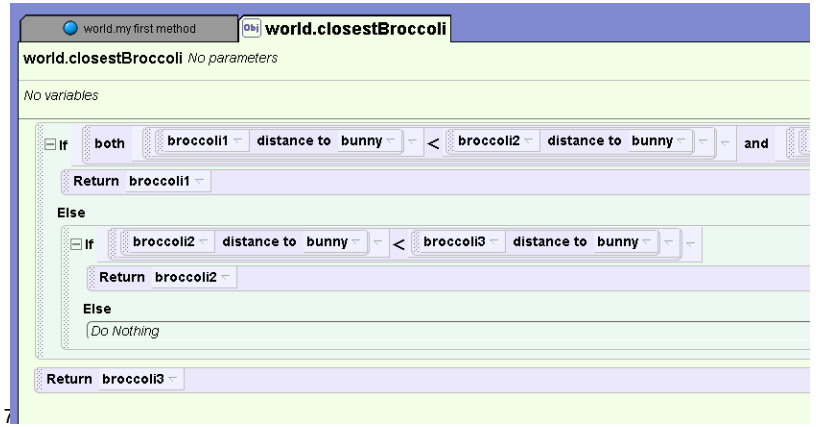
- replace the value after the and with the world level function *a<b* and then compare broccoli1's distance to the bunny to broccoli3's
- results in a long Boolean expression
- if true, return the broccoli1 object






# Checking Other Broccoli

- in the else, repeat for broccoli2
- make the last return broccoli3






# Calling closestBroccoli

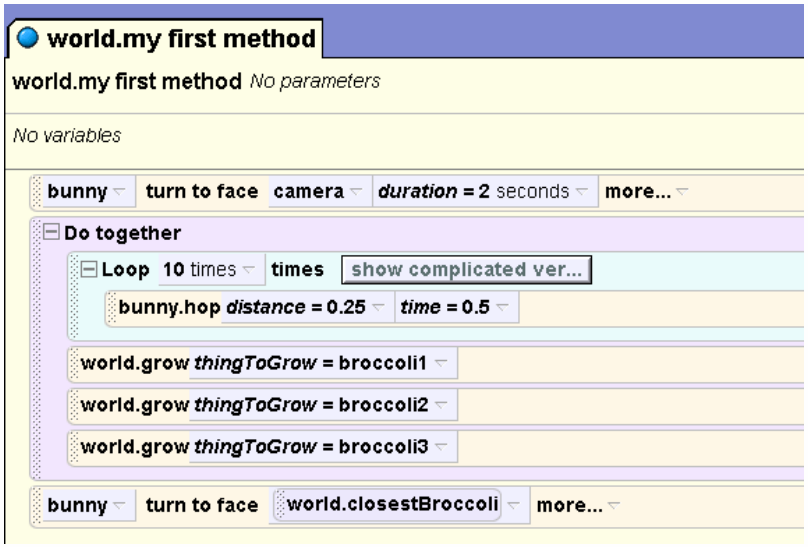
- Go back to my first method.
- Select the bunny and drag a turn to face command.
- Pick expressions and then the function closestBroccoli for the argument.




# Test

- Test function by playing movie
- Test further by changing initial set up of broccoli to change which broccoli is closest

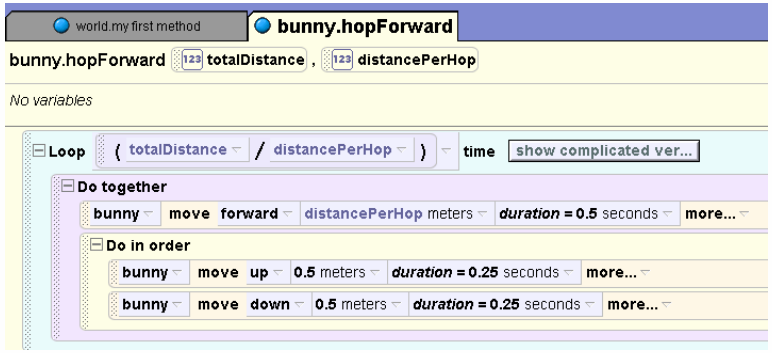
# my first method






# Hopping Forward

- Create a new method hopForward
- Parameters for total distance and distance per hop
  - lots of other ways to do this






# Completing the Hopping

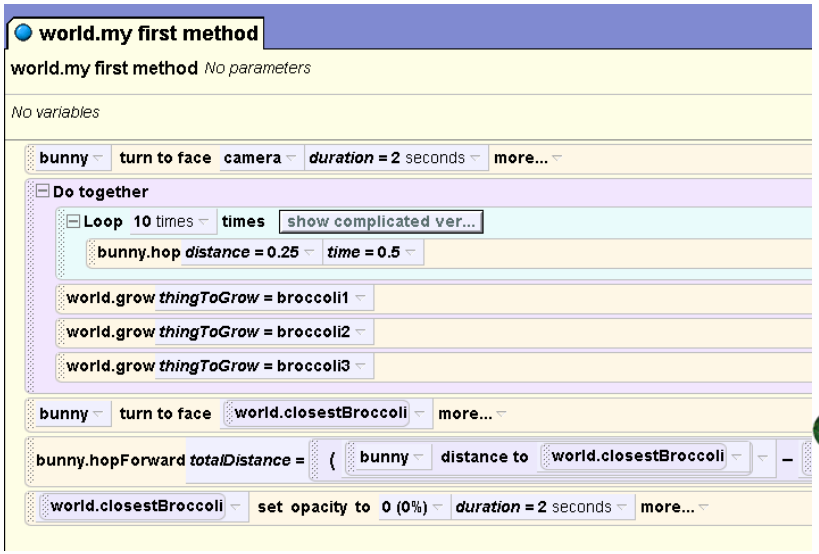
- Back in my first method call the hopForward method.
- Pick a dummy value for totalDistance.
- Replace dummy value with distance from bunny to closestBroccoli minus some offset. (no collision detection)




# Eating Broccoli

- Make closestBroccoli disappear
- Could add some motion to bunny

# Complete my first method






# What Next?

- Expand by adding more broccoli
  - lists and variables to manage
- Add sounds
- Add scenery
- Add events
  - Interactive programs can be created by adding events that program responds to.