Creating a Fancier Fox and Inheritance in Alice 3

Overview

- We will add procedures, functions, and properties to a fox and save it out to be used in other worlds.
- This is called “inheritance.” The modified fox has new things it can do, plus it inherits everything a standard fox can do.
- We will explore how Alice 3 uses classes to implement inheritance in another way.

Start – Add a Fox

- Select any background (I picked snow).
- Add a fox to your world (in the Quadruped folder).
- Choose the default fox. Click “edit code”.

Add a Fox Property

- Suppose our fox can only jump 1 meter, so we will code that in a property.
- Add a Fox Property
Jump Height Property
• Name it “jumpHeight”
• Set the value type to decimal number
• Set the initial value to 1.0 and click OK

Add a Fox Function
• We want a function to check our jumpHeight property
• Add a Fox Function

Check Jump Function
• Name it “checkJumpHeight”
• Set the return type to Boolean (True or False)
• Click OK

Writing checkJumpHeight
• Drag in an “If” block and select true as a placeholder
Writing checkJumpHeight (cont.)

• Change the condition of the if statement to check if the jumpHeight property is 1

Writing checkJumpHeight (cont.)

• Drag in two return statements
• Here is the final code

Adding a Fox Procedure - Jump

• We want to write a procedure to make the fox jump
• Add a Fox Procedure
• Name it “jump”

Writing jump

• Drag in an “if” block, select true as a placeholder
Writing jump (cont.)

- Click the Functions tab under the Fox
- Drag in checkJumpHeight over the true

Test Jump

- Click the myFirstMethod tab
- Drag in a “fox.jump” block
- Click “run,” and the fox should jump. What does it mean if it doesn’t?

Add an Elk to the World

- Click “Back to Scene”
- Click “Setup Scene”
- Add an Elk from the Quadruped folder.
- Position the Elk next to the fox.

Writing jump (cont.)

- Under the if, drag in a move up by jumpHeight and a move down by jumpHeight
- Under the else, drag in a say “I am supposed to be able to jump 1 meter, but I am not coded that way!”
Adding a Fox Procedure - goAround

- We want to write a procedure to have the fox circle the elk.
- Add a Fox Procedure
- Name it “goAround”

Writing goAround

- Add a parameter to represent the object to circle

objectToCircle Parameter

- Name the parameter “objectToCircle
- For value type, select “Gallery Class…”

objectToCircle Parameter

- In the middle window, select “SModel” and click OK.
- This structure will be discussed later!
Writing goAround (cont.)

• Drag in a turnToFace and select objectToCircle
• Drag in a turn, select left and 0.25
• Drag in a turn, select right and 1.0.
• Click “add detail” and select asSeenBy objectToCircle

Testing goAround

• Click on the myFirstMethod tab and drag in a “fox.goAround” block
• Select “elk”
• Click “run”

Now Save our Fancy Fox

• Click on the Fox tab
• Click “Save to Class File”
• Name it “fancyFox”

Fancy Fox in a New World

• Open a new Alice world with grass, setup scene
• Add a fancyFox from the My Classes tab
• We can see all of the information transferring. Click “Finish”
Add a Bear

• Add a bear to the world from the Quadruped folder
• Position the bear next to the fox

Test jump and goAround

• Go to the myFirstMethod tab and drag in “fox.jump” and “fox.goAround” bear.
• Click “Run”

Inheritance

• Inheritance is essentially when an object is derived from a parent and inherits the characteristics of that parent
• Our fancyFox inherited the characteristics of a regular fox
• Alice 3 showcases inheritance in another way too – classes!

Classes

• Remember this?
• Everything inherits from everything nested above itself.
• A Fox inherits from Quadruped, SQuadruped, SJointedModel, etc.
Classes (cont.)

• This means that we should be able to write a procedure for a Quadruped, and the Fox and Bear should both be able to do it
• Let’s test it out!
• Add a Quadruped Procedure

Writing gallop

• Create a new DecimalNumber parameter called “distance”
• Drag in a move, select forward, distance. Click “add detail” and set the animationStyle to BEGIN_AND_END_ABRUPTLY

Gallop Procedure

• Name the procedure “gallop”
• Drag in a “do together”

Writing gallop (cont.)

• In the dropdown, select “this.getFrontLeftShoulder”
• Drag in a move, and select down, and 1 as a placeholder.
• In the dropdown on the left, select “this.” Go to the functions tab, and drag in a “getHeight” block over the 1
• Select math and divide by 4.
• Set animation style to abrupt.
• Code on next page.
• Make a copy of the last line, and change it to “getFrontRightShoulder”

• Drag in a “this.move” block. Select up and 1 as a placeholder. Replace the 1 with the function: “this.getHeight.” Use the math dropdown to divide by 7

• Make 2 more copies of the last line. Make the following modifications:

  • Make a copy of the entire “do together” block.
  • In this copy:
    • Change every 4 to a 2.
    • Change every 3 to a 1.5.
    • Change every up to down.
    • Change every down to up.
  • Code on next slide.
• Make another copy of the first “do together” block.
• Remove the last “this.move” up block in this copy.
• The full code for gallop is on the next 2 slides.
Test it out!

• Put a block for “bear.run” and “fox.run” into your myFirstMethod and click run.
• Try it with other quadrupeds!
• Does it work with an object from the biped or flyer classes? Why?
• Only objects that inherit from quadrupeds will be able to access our gallop procedure.

Something to Remember

• Suppose you want to write a method in which an object’s opacity is changed.
• An SModel’s opacity can be changed
• But an SThing’s cannot
• There are other instances where complexity is added in subclasses that is not accessible to parent classes.