Lab 2 Etoys Basics

Lab 2 Etoys Basics

You already have used important programming concepts during the car tutorial. In this section, you'll get a better understanding of what actually happens and why - when you code your objects to do more interesting things.

Real world vs virtual world

Remember the virtual car? Well, everything in the programming world is virtual. The real world is the world around you – everything you see, feel or interact with is part of the real world. The virtual world is a representation of the real world – something that you, as a programmer, can create and control.

So, since we are programming in the virtual world, why even bother about the real world? Well, virtual world objects are very similar to real life objects. The real world has objects with properties and behaviors. The virtual world has objects with variables and scripts.

What is an object?

It is anything you want it to be. It is something that you draw, or that you pull out of the supplies bin in Etoys, or it is something that is composed of many smaller such parts. Writing programs in Etoys, or in many object-oriented programming languages, amounts to describing a collection of objects (the "things") and their behaviors.

Properties of objects

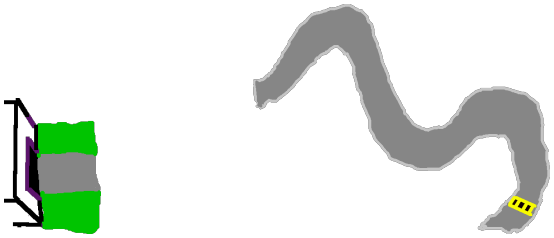
Each object has *variables* or *properties* that describe its basic attributes (such as height, width, length, color, speed (say, for cars), or height, hair-color, age, mood (for people)). These attributes can be constant (like a car's width), or may vary (like a person's age). Since in principle, any of these can be changed (varied), we call them all variables.

In addition to specifying *what* the objects are, we must also say *how they behave*, which is done by writing *scripts* in Etoys (called "methods" in Java). In the following, we'll learn additional programming constructs by creating a roadway for our car to drive on as part of a simple car-driving game.

Step 1: Recovering Your Work.

Open Etoys, and "Find" your project by clicking on the "Find" button on the Navigator Bar. Click on "Etoys" in the left pane, select your project in the right pane, and click "OK".

Either draw the following, or drag them from the desktop if they have been provided. Name them garage and road. Shown here, reduced and not to scale.



Lay them out so that the road connects to the garage.
Click on the car and the wheel to bring them to the front.

Just like other programs, you can usually drop a file into a document that is open by an application program that knows how to handle the file. Etoys knows how to handle image files – it imports them and pretends they are pictures you have just drawn.

No pane, no gain: Using panes in a viewer

Notice that the viewer is broken up into different "panes". Each pane shows a category of action tiles and properties. You may see "search", "basic", or others.

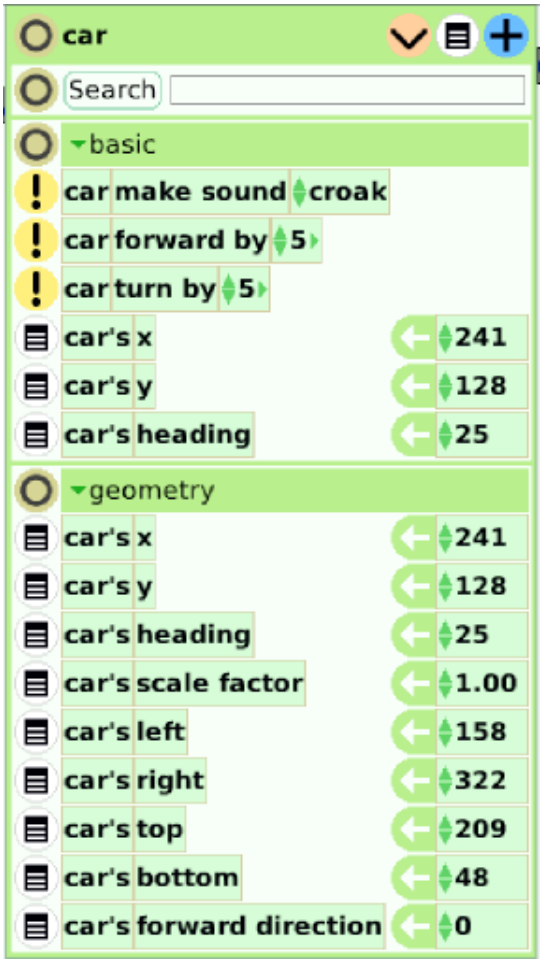
Click on the blue plus sign that is in the top right corner of a viewer. Notice how more "panes" appear? Click on a circle at the top of one of the panes, and notice that the pane disappears. Don't worry, you can get it back again.

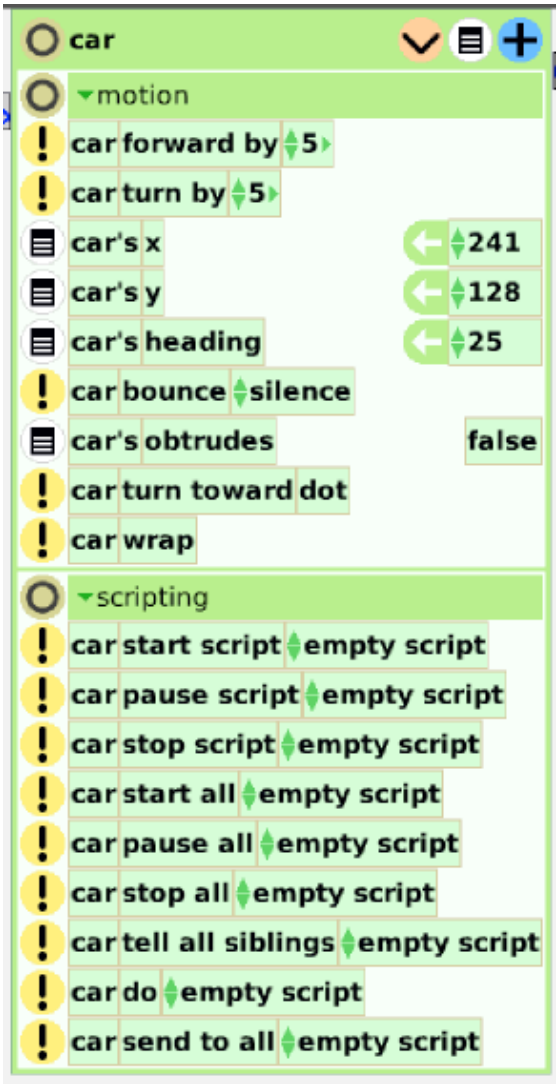
You can have as many panes open as can fit on your screen.

Every pane is the "same" in that it can hold any of the categories that you set it to. To change the category of tiles appearing in a pane, use the pull down menu at the top of the pane. On the next page, we've shown you some of the more useful panes, and the instructor may talk about these, or give you time to explore. Make notes next to the tiles as you learn about them.

Don't get overwhelmed with options or concern about not absorbing it all. The basic pane has perhaps all you need right now, but "geometry" and "motion" have some additional useful properties. We'll see more panes as we need them.

In programming languages, there are often MANY MANY options available. One of the skills of learning a language is slowly learning over time the various commands that you can use. Languages like Java have large libraries of pre-made scripts that you can import to your own program instead of doing it from scratch. But searching for the right thing can also be frustrating. For now, learn to use the tools that you know about to best advantage. When you learn a new command in a new pane, you may say "Oh… that makes doing such and such soooo much easier!"





Random numbers

Create a script to make the car react randomly when moused

Create a new car script called "jump" containing "car forward by 5".
Drag out a random number tile by first clicking on the icon of the small box near the right corner of the script header. Drop the tile over the "5" so that you get the following:

Fire it once by clicking on the exclamation point and see how much the car jumps forward.. Repeat and see how it changes each time.

"Random 180" picks a random number between 1 and 180 each time the statement is executed. The whole idea of randomness is that it is not always the same, and it is unpredictable.

Change "random 180" to "random 10". How would you make it jump backwards? Random 100 chooses a number between 1 and 100 randomly. Random –100 chooses a negative number between –1 and –100. How would you get it to jump EITHER forward or backward?

Hint: Start over with car forward by –5 and click on right arrow, and add random tile, to get car forward by –6 + random 11. Why does that work? List all possibilities.

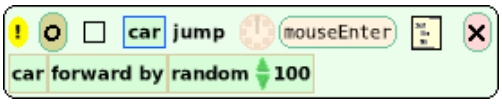
*You can choose the range of a random number. By using simple arithmetic you can change the range in interesting ways. What will [2 * random 10] result in?*

Random numbers are used in many ways in computer science:

- *For video games, you want some unpredictability.*
- *Random numbers are used in cryptography and in e-commerce to make sure that people can't figure out your Visa card, or impersonate you electronically.*
- *Random numbers are also used a lot in building computer experiments and simulations to see, say, how a disease spreads throughout a population, or how traffic congestion changes depending on how traffic lights are placed.*

Execute script on mouseover

Click and hold on "normal" and select a condition like "on mouseup" or "mouseEnter". See what happens when you click or mouse over the car.



Programming languages allow different ways for the programs to interact with the user. Sometimes you want to click, sometimes to just move the mouse over, sometimes you want to use the keyboard as an input device. A good programming language gives you ways to "listen to" various input devices, and use this information within a program. In a later lab, we'll be showing you how to control objects using the keyboard

Assignment Statements.

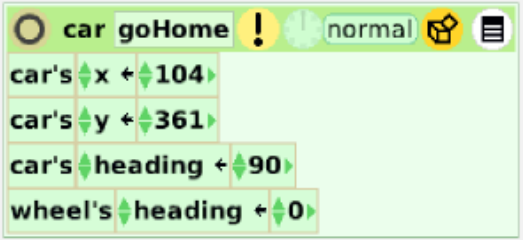
In Etoys, green/white arrows are "assignment" statements.
These can be used to change properties, like location, by programming.
You get them by dragging the *arrow* out of the viewer pane.

Assignment statements are one of the main tools in all programming. You store data by "assigning" its value to a memory location, much as you would in a calculator. You can later change the value by adding to it, multiplying it, etc, and "reassigning" its value to the new value.

Create a fire-once script to "reset" an object with assignment statements.

We'll create a script to position the car on the driveway ready to go

Drag out an emptyscript for the car, and call it goHome.
Drag green/white arrow car's x <- , and car's y <-, and car's heading <-
Click and see what happens.
Change the values on right side of arrows, and click ! again and see.
Can you figure out what the numbers should be so the car is in the driveway ready to drive out in the correct direction?
(hint: move the car to where you want it to be and see what the correct x, y coordinates are)
(warning: once the car is at the garage, it may be "under" the garage and hidden from view. If so, you may need to "send to back" for garage. this is done by selecting that option from the white halo icon of the garage *ask your instructor for help*)



Often, programs need to be "initialized" or "reset" to initial positions, or values for all data, before they can begin again. In a video game, that would be the player's health, level, location, etc. For medical diagnosis software, you might "zero out" the current patient data and start over, entering new data about a new patient. We wouldn't want to forget to set things back to "healthy" or "unknown", otherwise a new patient may look like she has the previous patient's measurements.

Create a button to fire a "fire-once" script

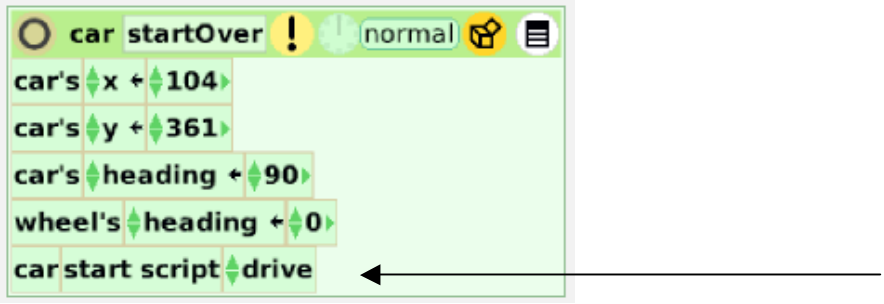
Click on the white menu in the top right of the goHome scriptor. Click and scroll down to select "button to fire this script". Set the button down where you'd like it, and try it. Note you can't move a button by clicking on it… you need to alt-click to get the halo, and use the black tongs or brown move icons on the top of the halo)

Often we use or create "widgets" like buttons and sliders to make it easier for the user to interact with a program. A lot of study goes into which layouts, and behaviors of programs, make for good user experiences.

Make one script "start" another ticking

Insert the tile "car start script drive" at the end of the "goHome" script.

The tile can be found in the "scripting" (not "scripts") category of the car's viewer. (now "goHome" is a bad name. Perhaps "startOver" is better. Change the name of the script to "startOver" by clicking on the name and typing the new name when it is highlighted. Make sure to hit "enter". Notice that your "goHome" button has now also changed its name automatically)



You should have something like the above script. Now, when you click on the button, the car should reposition itself in the driveway, and *start moving*. It's up to you to get to the finish line.

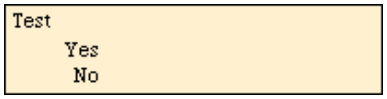
One script starting or firing another is a common practice in programming. If one big script, or program had to do everything, nobody would understand it. Each script is a "specialist" that handles just one part of the overall design. "startOver" doesn't need to know how the car's script "drive" works. It just tells the car to do its thing, and relies on the assumption that the drive script works properly. Just as large complex companies have specialists and an organizational hierarchy, and just as buildings have different subsystems (plumbing, electrical, roofing, …) software is broken up into subsystems and modular pieces that all work together. A challenge is to make sure that they DO work together, and not at odds with each other.

TESTS

We put the car on road, and test when it reaches the finish line.

Create a new empty script called "finishRace"

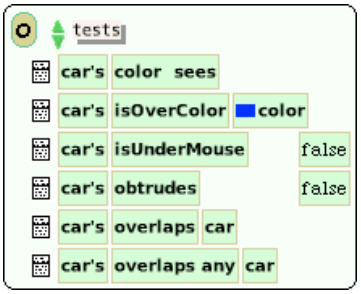
Tearing off a "TEST" box and placing into a new script

obtain this tile: 

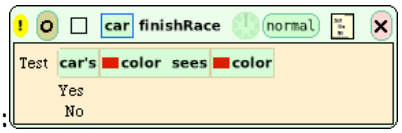
from here

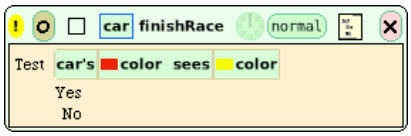
and place it in the script:

Obtaining "test" tiles from the tests category pane.
Your instructors will show you how to select the "car's color sees" tile, and go through the following sequence by clicking on the color boxes to set the test to see if the specific red of the car ever touches the specific yellow of the finish line.

Start here: 

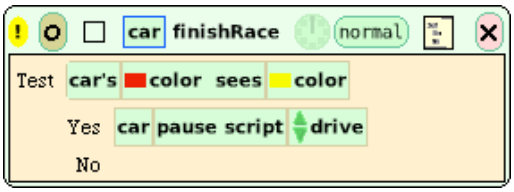
in the car's viewer ("tests" category pane) and drag and drop the top tile "car's color sees" into the finishRace script, and set the colors to get

first this: 

then this: 

We want to test when the red of the car touches the yellow of the finish line. We can set the colors in the "color sees" test by clicking on the first small colored box in the colorsees tile, which brings up an eyedropper, which we then click on the red part of the car. We do the same with the second box and the yellow of the finish line.

Finally, go to the scripting category pane of the car's viewer, and pull out the tile that says "car pause script emptyscript", drop it next to "YES" (look for the green highlight to know where to drop it), and change "emptyscript" to read "drive". You should know have:



Creating new Variables (properties)

How can we keep track of the "gas" used by our car? If you explore all of the panes in the viewer for "car", you'll find all sorts of properties like "heading", "theta", "brightness under", all of which have to do with sketches on a computer screen. But none of which have anything to do with being a car. Wouldn't it be nice if there was a "gas" property, and as the car drove, the gas ran out?

What is great about programming in any language is that you can pretty much do whatever you'd like. We'll create a "gas" property. We'll start calling these "variables", because their values can change, or vary. Most programming languages use the term "variables".

To create a variable, click on the small "v" at the very top of the viewer.
A menu will come up asking you to name the variable. Type in the word "gas".
Afterwards, the **variables** pane will probably appear.

Using "watchers" for variables

No decent car neglects to allow you to see how much fuel you have remaining.
To the left of the "car's gas" tile is a small menu icon. Click on it, and then select "detailed watcher", and pull out something like this:

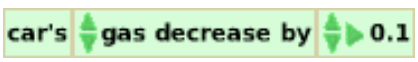
This is a "watcher", and is just a display that shows us the current value of the "gas" property. If you change the amount shown in the viewer, it will also change in the watcher, and vice-versa.

Let's start off with a lot of gas: Change the amount by typing in some large number, say 100.

Now, how can we make the car *use* gas as it goes, and, if it runs out, the car should stop?

You already know all the answers!

You can use a variable assignment to change the amount of gas at each tick, as the car moves forward. By clicking and dragging on the green/white arrow next to "car's gas" in the viewer, insert the following tile into the car's drive script.



By clicking on "gas increase"… "decrease"… , you can choose from several options. To enter a fractional (decimal) value as shown above for the amount of gas to "use" at each tick, simply type in "0.1" for example.

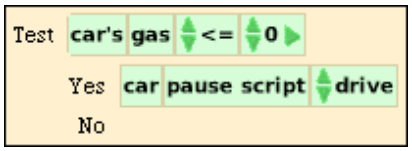
Testing the values of variables in TEST statements.

How can we make the car stop if we are out of gas? If we look on the "tests" pane for the car, we see only things like "overlaps" (if the car overlaps a particular other object), "colorsees" (very useful), but nothing that lets us test the "amount" of gas.

We'd like to test whether the gas is at 0, or (less than or equal to 0).

Every property can be tested to see whether its value is higher, lower, equal, to any particular number, or any other value.

Build the following, and add it to your drive script:



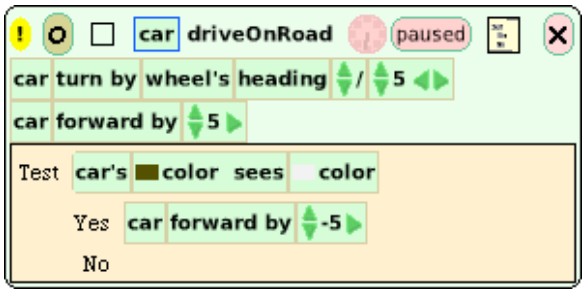
Where does the tile "car's gas <= 0" come from? If you drag out the variable name "car's gas" which you created, (*do not drag by the white arrow*), then when placed down correctly next to the TEST, it will expand to something like "car's gas < 5", and at that point you can change the number you're comparing it to, as well as whether the comparison is for less than, greater than, etc.

This tells Etoys to stop the "drive" script, which is the one moving the car, if the car should happen to run out of gas.

Now, see what the least amount of gas you can use in getting from the start to the finish!

Is there a way to stop the car should you run out of gas?

Extensions: Are you worried that people can cheat by going "off road". Use a "test" to keep the driver from escaping the roadway.



Tie all things together with start script, test for finish.

This is a good time to pause and think of three creative ways to extend the car project. Some examples are on the next page. Think of your own!

1. Place gas tanks on the road, if you go over them your gas increases

TEST car's overlaps gastank
YES car's gas increase by 10

2. Instead of gas being exhausted, why not record the time. Create a new variable called "time", set it to 0 in the "goHome" script, and then with each time tick increase the time by one.

3. If the car hits a banana peel, it skids out of control
Use a "colorsees" test over the banana peel, and the consequence could be a random turn and random forward move.

The possibilities are endless!

Lab 2 Skill Summary

Finding your project in the My Etoys folder
Using panes in a viewer: getting more panes, changing pane categories
    most useful categories: basic, motion, geometry, scripts, tests
Random numbers, and adjusting the random range, and random jumping
Execute scripts on mouseover and other events
Assignment statements give values to variables
Creating a "reset" script that does assignments
Buttons to fire a "fire-once" script.
One script starts another ticking
TESTS (conditional statements)
Obtaining test conditions from the **tests** category pane
    "colorsees"
One script pauses another's execution
Creating your own variables (example, "score")
Detailed (and simple) watchers for property and variable display
Use a test to keep the car from "cheating" by going off road

Lab 2 Review Questions

1. You and a friend are arguing over where to go to lunch. You have no coins to flip, but you do have Etoys. What can you do to decide the question fairly?

2. What pane would you go to to figure out how tall an object was?

3. What is the difference between "car start script move" and "car do script move"?

4. True or False: A "watcher" can be used to see the value of a variable or property of an object, but it cannot be used to change the value.

5. What does the test "colorsees" do? Give two examples of projects where you might use it.

6. What does the test "obtrudes" do? (You may need to explore this one on your own. Try dragging out a watcher for it, and then move your object around the screen, especially near the edges.) How can you use this in a game?

7. In the car tutorial, we forced the car to stop when it hit the edge of the road. How might you change the script so that the car turned back onto the road and kept on going?