Adventures in Alice Programming // Glossary of Useful Terms

by Michael Marion // July 2012

Worlds, methods, functions, properties, variables - it's pretty easy to get all of the terminology and concept matter confused when it comes to programming in Alice. To that point, we've developed this handy reference to use. Hopefully it helps you not to get mixed up. Enjoy!

World

A *world* in Alice encompasses everything. Just as we'd say that everyone and everything exists in our *world* in real life; every object, method, event, or anything else lies inside the *world* in Alice. The whole .a2w file represents the *world*.

Object

An *object* is just what it sounds like. Any sort of three-dimensional shape such as a person, animal, piece of furniture, building, or anything else of the sort in your world is an object. Most objects are added to the world via the *Add Objects* button, but some - the ground, camera, and light, for instance - are there by default. There are also two-dimensional objects such as text and billboards.

Properties

An object's *properties* are also what they sound like. All objects have certain properties - whether or not they're showing, their color, their position, and so forth.

Color

Defines the color of an object. There is a limited number of color choices in Alice, but it covers all primary colors and some extras, including red, blue, green, yellow, black, and cyan. To specify a white object, select "no color".

Opacity

Defines the transparency of an object. Unlike isShowing, setting the opacity to a middle amount allows it to be only slightly transparent. 0% opacity renders an object invisible, while 100% opacity renders an object completely opaque. When putting the Opacity property in a method, setting a duration will cause the object to gradually fade in or out of the scene.

isShowing

Determines whether or not the object is visible. A less gradual transition than changing the opacity setting.

Vehicle

Establishes a one-way relationship between this object and the vehicle object. Setting an object's vehicle means that an object will remain the same distance and orientation away from its vehicle. If the vehicle moves, the object moves the same distance and direction. If the object is told to move, the vehicle does not move with it.

Skin Texture

Allows the user to wrap a two-dimensional image around the object. Some objects, such as the ground, have built-in textures - the ground's textures include grass, dirt, snow, sand, space, and water.

Filling Style

Seldom used in Alice. Allows the user to render the object as a solid, wireframe, or points. Try playing around with this to see what it does!

Class

A *class* could be described as a template or blueprint for an object. All objects of the "Chicken" class, for instance, have the shape of a chicken, the color of a chicken, and can do the all the same things as a chicken. When we add an object to our world, we're creating an *instance* of that class - we're using the blueprint to make objects of that type.

Method

A *method* is a series of instructions given to objects that describe actions to take. All objects in Alice have some methods already built-in. The vast majority of objects can "say", or "move", or "turn", and so forth. However, we can write our own methods for objects that are built from smaller ones. This is one of the fundamental ideas in computer programming - using smaller methods as building blocks for more complex commands.

For instance, let's say I wanted to make an object walk realistically in Alice. Walking is more than just moving forward, right? I would write smaller methods that have the legs bend at the knees and move one at a time. I could write other methods that make the arms swing back and forth. I could then combine these methods into a larger "walk" method that would make the character walk like a human being.

Parameters

Methods can have *parameters*, which are essentially pieces of information that the method needs to execute. In fact, most built-in methods in Alice have parameters. When you tell an object to "move", for instance, you have to specify a distance, right? This distance is the parameter. There is one "move" method that applies to every object in Alice; by using a parameter, we can tell the object to move as far away as we want.

Variables

The world, objects, and methods can also have *variables*, which are essentially placeholders for any value that can change. Variables are often used to store particular pieces of information that we want to use later on. Variables can be made within objects by using the *properties* tab or within methods by clicking on *create new variable* in the top-right corner of the method editor.

Types of Variables

When we want to use a variable to store some value, we have to specify what type of value it is. Is it a number? Is it a true or false value? Is it a word or phrase? Whatever type it is can only store that certain type - a number variable can only store a number.

There are also different levels of variables.

World-Level Variables

Are created under the "world" object. They can be used across all other objects and methods.

Class-Level Variables

Are created by default within a specific class. Objects that are dragged in from that class all share the same variable(s). These variables can be used by all methods created under that object.

Method-Level Variables

Created by clicking "create new variable" in the top-right corner of the method editor. These are used only within a single method and cannot be shared across other methods.

Importing Images

You can import an image into Alice to use later as a texture for an object by going to "File" and clicking on "Import Image". Doing this will not cause any immediate change to your Alice world - it simply introduces the image to the Alice world's file system so that it can be used later on in the Alice world.

You can import an image by going to File in the top-left menu, selecting "Import Image", and choosing the image file on your computer. Imported images can be found by going to the world's properties and clicking on "Seldom Used Properties".

Billboards

Creating a billboard with a 2D image causes an immediate change to your world. It automatically creates a flat box and maps your image on it, making a "billboard" of your image that you can then manipulate in your world.

Events

By themselves, methods do not actually make objects do anything. Sure, you might know how to get to my house from Duke, but you wouldn't come to my house unless I called you and told you to visit me, right? We use *events* to *call* our methods. Events tell objects when to carry out their methods.

When The World Starts

Methods called by this event start as soon as you click "Play".

When A Key Is Typed

Methods called by this event start once you press a certain key. You can specify which key triggers the method.

When The Mouse Is Clicked On Something

Methods called by this event start when the mouse is clicked on an object. You can specify which object triggers the method when clicked.

While Something Is True

Methods called by this event will happen over and over again until the condition becomes false. You must specify the condition under which the event is triggered. *You can also right-click on this method to change it to "When Something Becomes True". Methods called by that event will occur once when the specified condition becomes true.*

When A Variable Changes

Similar to "When Something Becomes True", although this could be modified to encompass a variety of situations, such as when a boolean variable becomes false, when an object changes color, and so on.

Let The Mouse Move Objects

This event does not call or trigger any methods. Instead, it allows you to specify a list of objects and makes the objects in that list clickable and draggable by the mouse while the world is playing. *You must use a list with this event*.

Let The Arrow Keys Move Subject

This event does not call or trigger any methods. Instead, it allows you to specify a single object and allows you to use the arrow keys to move that subject around the world.

Let The Mouse Move The Camera

This event does not call or trigger any methods. Instead, it allows you to click and drag the view and move the camera around the world.

Let The Mouse Orient The Camera

This event does not call or trigger any methods. Instead it allows you to click and drag the view and move the camera's point of view from a fixed point. Use this method to manually look around your world while it plays.

Functions

Functions provide us with information about objects in our world. While properties also do this to an extent, functions generally calculate some sort of number, whereas properties are more qualitative - things like color, orientation, and so on. Like methods, we can use smaller functions to create bigger functions that calculate more complex numbers for us.

Note that examining the *world*'s functions in the object tree will give you very generic functions that can be used in a variety of applications.

Lists

A collection of items in Alice. Lists are typically not strictly ordered, meaning that items are sometimes not necessarily presented in the order they were added to the list. When you process a list, you do something to every item in the list. Lists use the "For All Together" and "For All In Order" tags as well.

Arrays

A collection of items in Alice. Arrays are more strictly ordered than lists, and individual elements can be accessed by themselves if need be. Arrays, however, do not work with the "For All Together" and "For All In Order" tags. To process elements in an array, use the "Loop" tag.

Tags at the Bottom

If you glance at the bottom of the method editor, there are different colored tags with labels such as "Do In Order", "Do Together", "If/Else", "Loop", and so forth. Dragging a tag into the method editor will create a small block into which statements may be placed. The labels work as such:

Do In Order

Statements put inside this block will execute in order, one at a time, until the statement is completed.

Do Together

Statements put inside this block will execute simultaneously. Note that statements after the Do Together block will not execute until every statement in the Do Together has completed. Statements within the Do Together of longer durations will still take longer to execute than their more brief counterparts.

If/Else

Allows a condition to be specified. If the condition evaluates to "true", the first segment of code will execute. If the condition evaluates to "false", the second segment of code will execute.

Loop

A block of looping code will repeat itself as many times as is specified when it is dropped in. Be advised that infinite loops are dangerous and can crash your program.

While

Similar to a regular loop - a while loop will continue to repeat itself as long as the specified condition evaluates to "true".

For All In Order

Works specifically with lists in Alice. The block of code will execute for each item in the list, one item at a time in order.

For All Together

Works specifically with lists in Alice. The block of code will execute simultaneously for each item in the list.

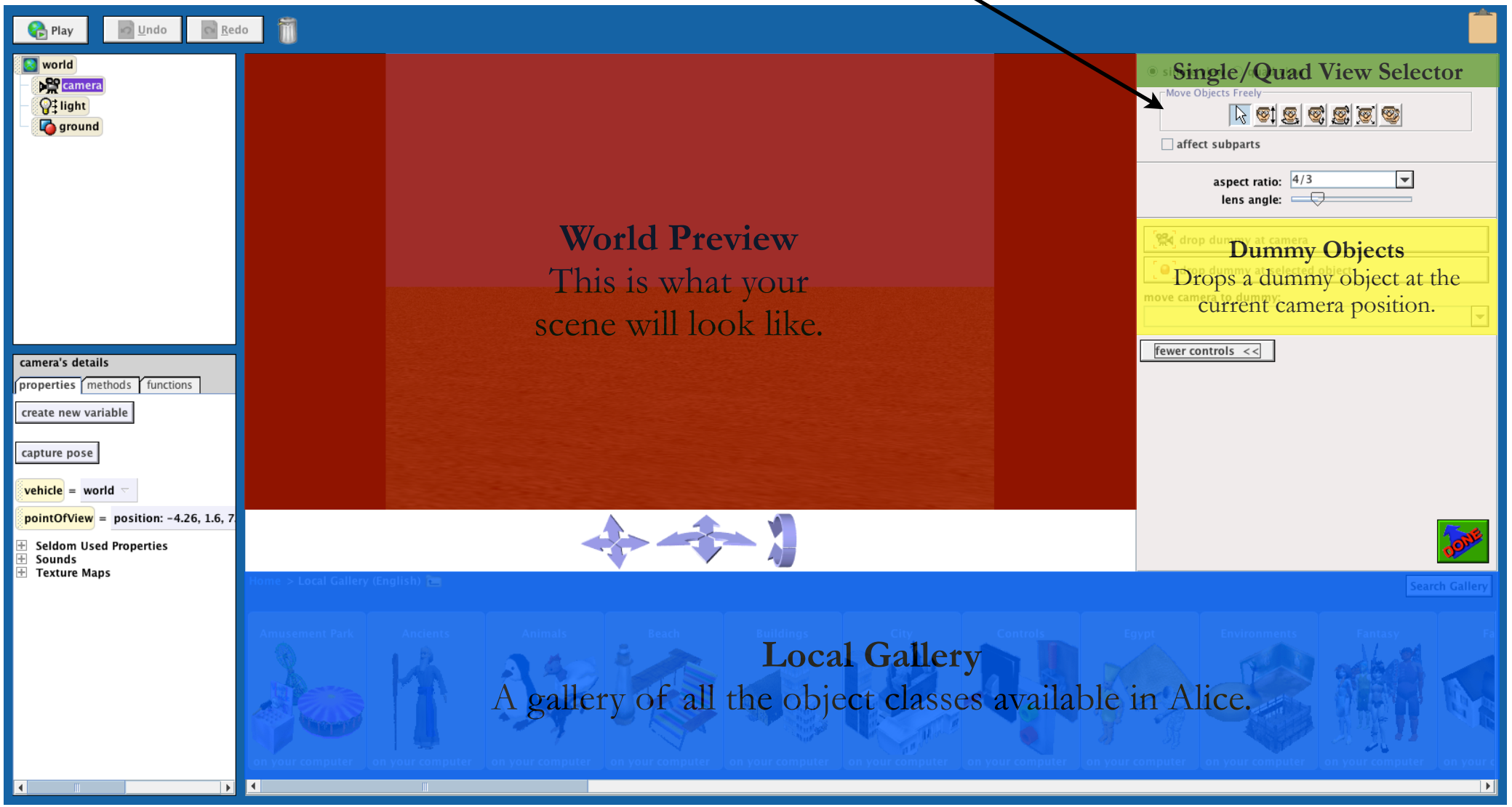
Poses

Poses are just what they sound like. Poses allow you to save particular stances of an object to quickly toggle among them. For instance, say I wanted to make someone dance the "Y-M-C-A". Instead of writing complicated methods that move their arms and head around, I could position the character in each pose and save each pose separately. Then I can tell the character to set its pose to each letter in the dance and it will automatically move gracefully between the poses.

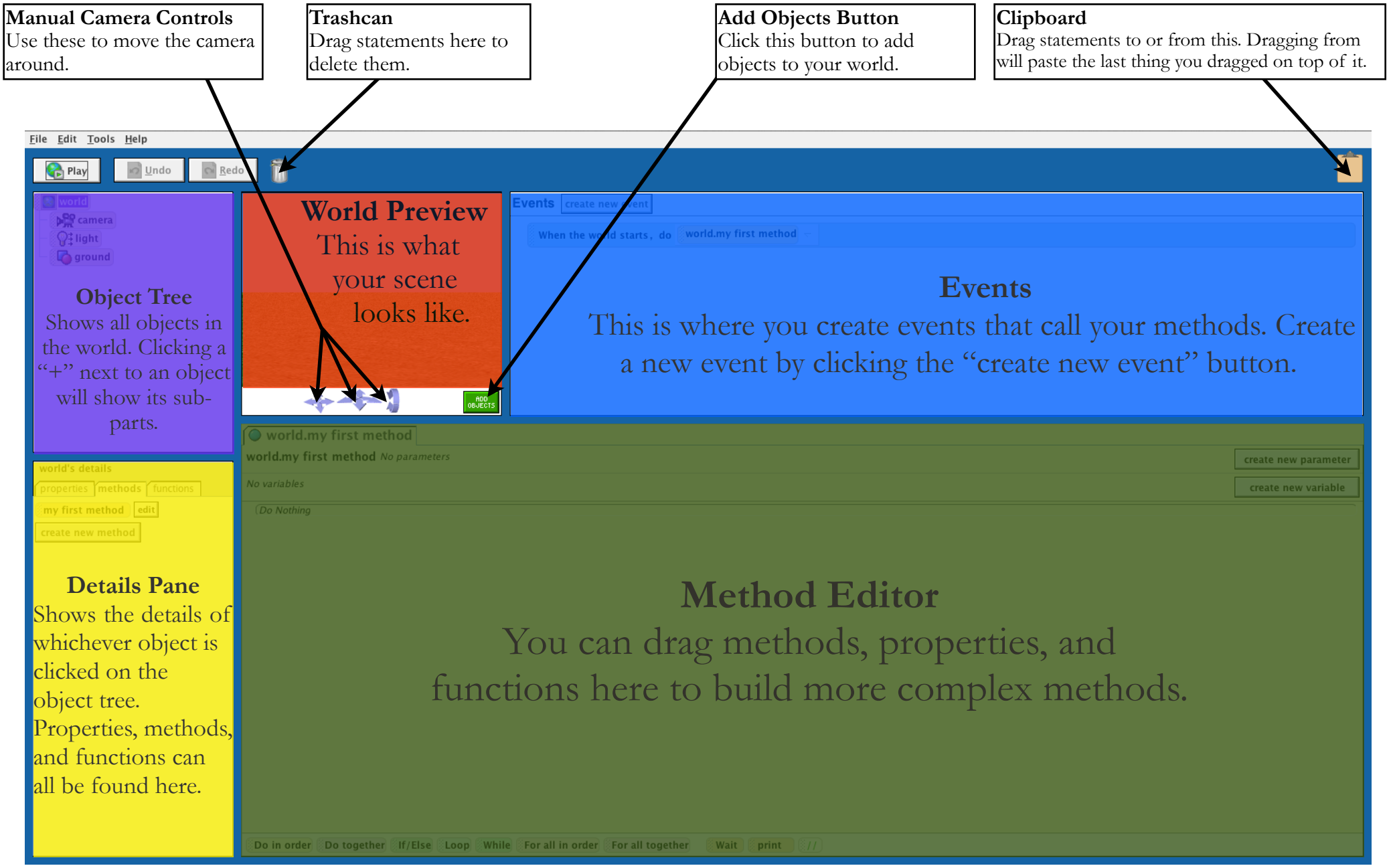
Comments

Comments allow you to write lines of code that don't execute. They are what they sound like: adding comments to your code allows you to create quick reminders or plain-English remarks on the function of your code. Comments are more for the user rather than the computer.

Switch Object Movement
Use these buttons to restrict the movement of objects when you click to drag them around.



Layout of the "Add Objects" Panel



Layout of the Alice Programming Environment