Note: In this tutorial, you will only see full screenshots of the Alice environment when you have to move things across the screen. If you need to know where each part of the Alice interface is, please examine the screenshots in previous tutorials.

-1. Intro to the World

The game, as it currently is:
This world has mostly been pre-built and we will only be making a few changes.
The world is shrouded in darkness, and only by collecting 6 of the magical energy cubes, our mighty hero can restore light to the world.
Use the T key to teleport, and the F key to face towards the closest gem.
Download the world here.

Your Task:
Currently, the game can be won by using the F key several times and then just running forward until you arrive at a gem. Also, the game does not do anything when you've collected all of the gems. Your task is to create two class-level variables to limit the wizard's power so he cannot simply use the F key five times and win. Also, make it so that, when he collects 6 gems, the game winning animation shows up.

0. When is it appropriate, and why?

In a previous tutorial, the use of variables as state storage was introduced. Here, we will introduce the concept of class-level variables, which are in variables that are tied to a particular object in the Alice world. We use such constructs when it seems "logical" in the context of our creation. For example, a wizard has magical energy, a warrior has a limited amount of health, and a character has a name. Anytime you can think of a quality as being "part of" an object, make it a class-level variable.

1. Let's Get Started: Creating the Variables

Load the world, and hit Play. Notice that the scene you see when you open the world is different from what you see when you play the world. Now, we are going to start by giving the wizard mana. Click on wizard in the object tree and click properties. Then, click on "click new variable."

In the dialog box that appears, I have created a variable called "Mana (Magical Energy)" of type number, and set its initial value to 100. You
can name your variable anything you like; I choose the name so that programmers who aren't familiar with the term "mana" will have some idea of what that variable is supposed to represent. Also, I like to think of started at 100 mana, but you could just as easily start your wizard with 1 mana, and work your way down in fractional decrements.

Now, we want to create one more variable within the wizard to keep track of how many gems he has collected, so we can eventually make a condition for winning the game. This variable will also be of type number, since it keeps track of how many gems we have collected. (Note: If all the gems were of different types and each had special powers, then we would instead create a list of objects and probably call it something like Inventory.)

Next, we've given the wizard mana, but it would seem reasonable for him to actually EXPEND his mana when he does magic. After all, nothing is free. Our wizard can perform two acts of magic at this time (you are welcome to expand my game and add more), so let's start with the Teleport method of the wizard. For the purpose of this tutorial, my wizard is going to expend 10 mana every time he teleports, so I want the mana variable to go down by ten every time this method is called. Open the method, then open the properties of the wizard.

Note that this variable, by the logic of the game, starts out as 0, since the game starts with the wizard possessing no gems, although we adjust this initial value depending on how we want the game to work. For example, we could have it that the wizard was able to hold onto one of the gems before it flew away, in which case the wizard would start with gemCount = 1.

2. Magical Efforts Drain Magical Energy: Accounting for Energy Used

At this point, we've given the wizard mana, but it would seem reasonable for him to actually EXPEND his mana when he does magic. After all, nothing is free. Our wizard can perform two acts of magic at this time (you are welcome to expand my game and add more), so let's start with the Teleport method of the wizard. For the purpose of this tutorial, my wizard is going to expend 10 mana every time he teleports, so I want the mana variable to go down by ten every time this method is called. Open the method, then open the properties of the wizard.
Now, to increment, we want to drag the Mana variable from the properties tab, and drop it as the first line of the method Teleport. Then, select "expressions" and choose one of the (potentially) 2 Mana variables (which are identical - the fact that there are two is a bug in Alice).
Now, by itself, this statement seems to do absolutely nothing, and if you thought so, you are absolutely correct. Click the down arrow next to the select Mana variable and select Math -> Mana --> Other. In the box that appears, type 10 and click Okay. This will set the current mana level to ten less than the current mana.
level, each time the method is called. Similarly, we would like to change the world.locate method so that it cost 30 mana for the wizard to use it since it is almost like a magic bullet in the context of the game. When you are done, your "locate" method should look like this:

Now, we turn to the other variable.

3. Gems Need not be Found Twice: Keeping Count

If you look in the events pane, you'll notice I have already provided you with a method that says "checkForIntersection." Open this method and look at the current code.

Observe that this method executes an infinite loop, which constantly loops through all eight of the cubes, checking for intersection. Currently, at an intersection between the wizard and the cube, the cube goes out of sight. Now, we'd like to add some code so that the wizard keeps track of how many gems he has collected. To do this, we can drag GemCount into the method and choose "Increment wizard.GemCount by 1".
Now, each time there is an intersection, the wizard's gemcount will increase by one, and Alice will therefore know how many have been collected thus far.

4. Keeping Track is Not Enough: Implementing Conditions Based on Class-Level Variables

At this point, our variables are changing as they should be, but there is nothing to prevent the wizard from continuing to cast his spells with negative mana, and we can't win the game no matter how many gems we collect. Let's begin by restricting mana use, so that the wizard cannot cast a spell if he has insufficient mana. Let's start with the teleport method. Open up this method, and drag an if statement into it, selecting True temporarily.

Then, as described in the "Constructing an Expression" Tutorial, we want to make the if statement say "If mana ≥ 10." When you are done with this step, your screen should look like this:

Now, we want the teleport code to only be executed if this condition is true, so we want to drag all of that code under the If statement. Unfortunately, the present version of Alice does not allow the selection of multiple statements, so you will have to do this one at a time, and make sure you preserve the ordering.
When you are done, your teleport method should look like this:

```
wizard.Teleport No parameters

if wizard.Mana (Magical Energy) >= 10

    x = set value to ask user for a number question = How far would you like to move?

    Z = set value to ask user for a number question = How far would you like to move?

    camera = set vehicle to world

    Do together
        world = set ambientLightColor
        wizard = set emissiveColor

    Wait 2 seconds

    Do together
        light = set brightness to 0
        wizard = set emissiveColor

    Wait 2 seconds

else
    Do Nothing
```

Now, go through the same steps with the "locate" method of the wizard.
Finally, we need to make it so that, when you collect 6 gems (that is, GemCount gets to 6, the winning sequence is displayed (this is already created under the world method "winner") For this, we must realize that the natural place to check for a winner is immediately after the gem count has been incremented (think about why this makes sense). That is, we need to add to the code under the infinite loop of the checkForIntersects method. Open the checkForIntersects method and add an If-statement right after the "increment" statement.

Then, replace the "true" in the If-statement with the condition that the gemCount = 6, and drag the world method winner under the If-statement. When you are done, you should see something like this:
Now, when you play the game, it should be functional, in the sense that there is a limit to the number of times our wizard can use his magic, and, when he does manage to collect 6 gems, he is a winner.

5. The Player Wants To Know: Adding displays for Mana and Gems Collected (Optional)

Now, there are a couple of embellishments we can make to our game. For example, we could add a feature so the wizard shrieks (okay, says), "I don't have the mana for that!" every time you try some magical act and lack the mana for it. In this tutorial, however, we will just demonstrate how to update the display of the class-level variable to the user, in the form of 3D text. Since the point of this exercise is not to demonstrate object placement, here is a file with the 3D text already in place. We will simply go over how to code it. Let's start with the Gems, since the gem count is only changed in one place. That place is in the method world.checkForIntersection. Open this method and look closely at it:

Notice where we increment the GemCount. I hope it seems logical to the reader that we should update the display right after we've collected a new gem:
In order to update the display, we want to update the "text" property of the "Gems Collected" label. Click on "Gems Collected" in the Object tree, click the properties tab, and drag the "text" property directly into the code after the "increment" tag. In the drop-down menu, choose "Default String" (we will replace this).
Now, we need to change the text to a combination of two things: the word "Gems" and the current number of gems. Go to world's functions, and drag the "a joined with b" template to replace the default String. Again, choose something random at the drop-down menu.

After you drop, you should see something like this:

```
world.checkForIntersection
```

```
world.checkForIntersection (no parameters)

No variables
```

```
Loop infinity times times show complicated version
```

```
Loop 8 times times show complicated version
```

```
both Item index from world.cubes isShowing and
```

```
Item index from world.cubes set isShowing to false more
```

```
Gems Collected set text to a joined with b more
```

```
if wizard.GemCount == 6
```

```
world.winner
```

```
else
```

```
Do Nothing
```

```
else
```

```
Do Nothing
```

```
Wait
```

```

```
Do in order Do together If/Then Loop While For all in order For all together Wait
```

```
Now, click on the arrow in the left half of the template and select "other." Type "Gems: " in the box and click ok.

Now, for the second half of the template, we want to drag the template "What as a string" from the world's functions, because we are going to convert the Number variable, GemCount to a string. When you drop the "What as a string," you will be prompted to choose something. Select GemCount.
Selecting GemCount:

Completed State:

Now, when you collect a gem, the GemCount variable should be updated, and the display should be similarly updated. Please try on your own to do the same with the two places where the mana amount is changed. When you are done, the teleport and locate methods should look like the following.
Now, your world should have functional displays for both mana and gemCount. Play your world.

This concludes the Class-Level Variables Tutorial. We have covered how to create class-level variables, the types of things they are useful for representing, and (optionally) how to display them to the user in a game or animation setting. If you have any comments, please email me.