Wizard Game:  
Class-Level Variables in Alice

By Jenna Hayes  
Under the direction of Professor Susan Rodger  
Duke University, July 2008  
www.cs.duke.edu/csed/alice/aliceInSchools

Introduction – Playing the Game

The point of the game is to help the wizard collect the 6 lighted cubes that will hide themselves at the beginning of the game:

Press F to make the wizard face towards the nearest cube.  
Press T to teleport to a different location.  

Your task is to limit the wizard’s power so you can’t just press F six times to win. You also need to make it so that when the wizard collects all 6 cubes, the game winning animation shows up letting the player know he/she has won.

Download the starting Alice World that goes along with this tutorial. You will modifying the world to create a wizard game that keeps track of the wizard’s magic as well as when the player wins.

Variables – Class Level Variables

Try playing the world to see what happens and familiarize yourself with how the game looks. The opening animation should play in which the lighted cubes fly off to different places in the world.

To change this world we are going to create something in Alice called class level variables. We will create a variable for mana and a variable for gems. Then we are going to display them as the game runs so the user knows how much of each there is.

Variables – Creation

We are going to give the wizard mana, which in typical role-playing video games are points that a character can use to perform magic. Spells cost mana and when mana runs out, the wizard can no longer use teleport or find nearest cube.

Click on the wizard in the object tree, and then in the properties pane, click create new variable.

Click on the create new variable button labeled "Create new variable". Name your variable Mana (Magical Energy), Make sure that it is a number variable, and set its initial value to 100.

A variable is a space where information that changes can be stored, such as a number that needs to be added to or subtracted from as a game is played.

A class-level variable is information that changes and is tied to a specific object, as opposed to just general information.
Variables – Creation

Notice in the properties pane that the variable has been created. Now create another variable to store the number of gems that the wizard has collected so far. This will help so we can create a condition that we can check so we know when the game has been won. Call it GemCount and set its starting value at 0.

Now we want to make it so that when the wizard does magic (either teleport or locate) his mana goes down. Click on the wizard in the object tree. Go to methods and click edit next to the Teleport method. This will open up the method in the editor.

Go back to the properties pane and drag-and-drop the Mana (Magical Energy) variable to the top of the Teleport method. Then select set value, expressions and wizard.Mana(Magical Energy).

Click the down arrow next to the second wizard.Mana (Magical Energy) and then select math, and then Mana (Magical Energy) - and then other… On the calculator that pops up, enter in 10, and then click Okay. This will take away 10 from your Mana whenever the Transport method is called.
Now repeat the same process for the wizard's `locate` method. This time, instead of subtracting 10 from the mana, subtract 30.

This will subtract 30 from your mana whenever you press F to use the `locate` method.

Now we need to restrict the wizard so that he can only use mana if his mana variable has enough. We need to make it so that when he runs out, he can no longer do Teleport or Locate. Go back to the Teleport method, and drag an `If Else` statement to the top of it, selecting `true`.

Now, we only want him to teleport if his `Mana` is greater than or equal to 10, because that's how much Mana it takes to teleport. For this, we need a `>=` sign. Click on `world` in the object tree and then go to the `functions` pane. Find `a>=b` under `math` and drop it over the `true` on your `if` statement.

On the menu that pops up, select `expressions`, then `wizard.Mana(Magical Energy)`, then 10.

Now, we only want `Teleport` to work if the `mana` is greater than 10. This means we have to `drag and drop` all of the code in the rest of the method into the top of the `if` portion of the `if/else` statement. You will have to drag each piece one at a time, making sure to keep them in the right order. See the next slide to see what your `Teleport` method will look like.
Mana Restrictions – Teleport

Now do the same steps with your Locate method. In an If Else statement, make the statement If wizard.Mana(Magical Energy)>=30, and drag all of the code in the method into it.

Mana Restrictions – Locate

Now we want the GemCount to increase whenever a gem is found. Click on world in the object tree, and then go to the methods pane. Click on edit next to the checkForIntersection method.

Gem Coding

Drag the GemCount variable (in the wizard's properties) into the If Else statement and drop it at the end of the if statement (before else) there. Then select Increment wizard.GemCount by 1.
Gem Coding – End Game

Now we need to make it so that when all 6 gems are collected, the winning sequence plays (located in world.winner). Add an If Else statement right below the increment method.

Now go into the world’s methods pane, and drag and drop winner into your If Else statement in the If portion.

Now your game should be functional!

Now there’s only one problem: we can’t tell how much mana we have left, or how many gems we have collected, unless we keep track in our heads! We can fix this by adding text objects that display both of these numbers and update as the game is being played.
Now you need to move your camera position so you can see the text objects and the wizard. Right click on camera in your object tree, then select methods, then camera set point of view to, then Dummy Objects, then Starting Position.

Now use your object positioning buttons to push the text objects back from the camera, and then up in the sky, until they look approximately like this:

Click Done when you're finished.

Now click on Gems in the object tree and go to its properties pane. Find the small button that says vehicle. Click on the down arrow next to world, and select camera. Do this same thing for Mana. This will make it so that you can see the text no matter where the camera goes.

Now we need to put code in our checkForIntersection method that connects the GemCount variable to the Gems text object. First, make sure that you are looking at checkForIntersection in your method editor. Then click on Gems in the object tree, and go to its properties pane. Find the small button that says text. Drag and drop the text button into the checkForIntersection method right under the increment command. Select default string for now.
Click on `world` in the object tree and go to the `functions` pane. Find a `joined with b`, and drag and drop it onto your `set text to command` where it says `default string`. Just select `default string` again.

Now click on the small down arrow next to the first `default string` on your `set text to command`. Select `other...` and then type in "Gems:". Then click `OK`.

Now look back at the `world`’s `functions`. Find what as a `string` and drag and drop it on top of the second `default string` on the `set text to command`. Select `expressions`, and then `wizard.GemCount`.

Now open up the `Teleport` method, go to `Mana`’s `properties` pane, and repeat all of those steps for the `Mana`, typing in `Mana:` for the first `default string`, and selecting `wizard.Mana(Magical Energy)` for what as a `string`.
Now repeat those steps again for the `locate` method.

Variables – Display Mana

Now play the game. It should go through the opening as usual, except now there are two text objects on the screen also. Whenever you find a cube, your `gem count` should go up, and whenever you use either `Teleport` or `locate` by pressing `T` or `F`, your `Mana` should go down!

Testing the Game

Now try winning the game. Can you do it?
After you win, try winning without using `Mana`!