Checking for Collisions

Ellen Yuan
Under the direction of Professor Susan Rodger at Duke University
June 2014
Getting started

This tutorial will teach you how to write a function to constantly check for collisions between objects. You should already be familiar with using Alice.

- To get started, select the space world, and add some planets to your world – make sure that they are not too close to one another! I added all 9 to my world (Earth, Jupiter, Mars, Mercury, Neptune, Pluto, Saturn, Uranus, and Venus). The planets can be found in your Local Gallery in the Planets folder. Also add the cleanLabWorker to your world.
Creating a group

• To organize your object tree, create a folder in it by right clicking inside it and clicking on **create new group**. Right click on the folder and rename it to “planets”. Drag all the planets into the folder.
Dropping a Dummy Camera

- To be on the safe side, we always want to set a Dummy camera so that we can return to our original position. To drop a dummy camera, click on the green Add Objects button and then click on more controls. Select drop dummy at camera.
Renaming folders

• If you look in your object tree, you will see a folder labeled **Dummy Objects**. Let’s rename the folder by right clicking on it and renaming it “CameraViews.” Further, if you click on the plus sign next to **CameraViews**, you will see an object called **Dummy**. Rename that to “start”.
Creating a list

- We are going to create a list of all the planets. Click on the world’s properties and create a new variable called planet. We want to create an OBJECT list and add the number of planets you have – in this case, 9, but they will be named 0-8. Next assign each item to a planet.
Creating a function

• Our function is going to tell us whenever our cleanLabWorker collides with another planet. To do this, create new function called “collide” in our world’s function. We want it to be a Boolean function.
Writing a function

• We are going to use a **For all in order**. We want the function to go through each planet in our list, so drag it into your function editor and select **expressions**, and then **world.planet**.

• Drag an **If/Else** statement into your **For All In Order**, and select **true** for now.
Adding code

• This next part is going to be tricky. First we want to drag an a<b function and replace the true with it. Select 1 for both of them for now.
Adding more code

- Under the cleanLabWorker’s functions, drag `cleanLabWorker’s distance` to over the first 1, and then select `expressions`, and then `item_from_planet`. This if-statement will allow to check if the character is colliding with any of the other planets.
Even more code

- The objects are considered colliding if the distance from the cleanLabWorker to the planet is LESS THAN half of the cleanLabWorker’s width plus half of the planet’s width. To do this, we will use the math function. Drag the cleanLabWorker’s width to the second 1.
Using the math function

- To use the math function, click on the arrow next to the cleanLabWorker’s width, and scroll down to math, and then select cleanLabWorker’s width /, and then 2.
More math functions

- We want the distance of the cleanLabWorker’s to the planet to be less than the sum of the half of both of the objects’ width. Therefore, we need to add even more to our if-statement! Click on the arrow and select **math**, and then `(cleanLabWorker’s width/2) +`, and select **2** for now.
Finishing up the if-statement

- Now we are going to replicate the previous statement.
- Replace the 2 with `cleanLabWorker’s width`. Then, use the math function and select `cleanLabWorker’s width /`, and then 2. However, this time, replace `cleanLabWorker` with `item_from_planet`. 
Return statements

- If the distance between the cleanLabWorker to a planet is indeed less than the sum of half their widths, then they are colliding. Therefore, we want it to return true. Drag a Return under the if-statement and select true. In addition change the existing Return statement to false.
Finally done with our function!

• Your final code for the function should look like:

```plaintext
world.collide No parameters
No variables

For all world.planet, one item_from_planet at a time
If cleanLabWorker distance to item_from_planet < 0.001
  - (subject = cleanLabWorker's width / 2) + (subject = item_from_planet's width / 2)
  Return true
Else (Do Nothing)
Return false
```

• In the next few slides, we will create methods and events so that we can check for collisions and control the cleanLabWorker and have him move around space.
Creating a method

• How are we going to verify that our cleanLabWorker collided with a planet?
• Let’s made him jump back whenever he collides. Create a method for the cleanLabWorker by clicking on create new method, and name it “jumpBack.”
Adding the code

• Drag a **Do together** to your method editor and drag **cleanLabWorker move** into it. You can make your cleanLabWorker jump back however you want him to.

• I made him jump 0.5 meters in various directions, and I changed the **style** to **abruptly**, and made the **duration** last for 0.25 seconds. Click on **more** to see these options.
Final Code

• This is the code I wrote to make my cleanLabWorker to jump backwards when he collides with a planet.
Creating another method

- Now let’s edit our world.my first method.
- In your world.my first method, drag and If/Else statement and select true for now.
- Under your world’s functions, drag the collide function over to the true.
Finishing our method

• Next, let’s make the cleanLabWorker say “ouch!” every time he collides. Under his methods, drag `cleanLabWorker say` under the if-statement, and select other, and type in “ouch!” Then drag the cleanLabWorker’s method `jumpBack` below that statement.
Final code for world.my first method

- Here is the final code for your world.my first method

```plaintext
If world.collide
  Do together
    cleanLabWorker say ouch! more...
    cleanLabWorker.jumpBack
Else
  Do Nothing
```

Creating events

• We are going to create 3 events:
  – An event that moves the cleanLabWorker around
  – An event that moves the cleanLabWorker up
  – An event that moves the cleanLabWorker down
• Let’s have the arrow keys control the cleanLabWorker, so **create a new event** for that, and change the camera to the **entire cleanLabWorker**.
More events

• We want our cleanLabWorker to be able to collide with all the planets, but it needs some controls to move up and down. Let’s create events where, when the U is pressed, the cleanLabWorker will move up ½ meter, and when the D is pressed, he will move down ½ meter.

• Click on create new event, and then When a Key is typed, and changed any letter to letters, and then select U.
Finish writing the events

• Click on cleanLabWorker in your object tree and go to his methods. Drag a cleanLabWorker move over the Nothing in your event, and select up, and then ½ meter.

• Create another event so that when the D is pressed, the cleanLabWorker will move down ½ meter.
One more step!

• One last thing we want to do is change the When the world starts. Right click on it and select change to, and then While the world is running. This way, we will constantly be checking for collisions. Select `world.my first method` for the During: event.
Final code for events

• Below is a list of all the events you should have for the world
Play your world!

ONE LAST THING: if you would like, set the camera’s point of view to the cleanLabWorker, and change the camera’s vehicle (found in the camera’s properties) to the cleanLabWorker. This way, you will never lose sight of him. And you will always have the starting point because of your dummy camera.