Checking for Collision

by Vicki Zhang
Under the direction of
Professor Susan Rodger
Duke University
July 2017

The goal of this mini-game is to guide the floating buddy safely to the coconut. Therefore, two detections are involved:
1. Ghost and Coconut
2. Ghost and Fire

Collision

There are two ways of coding collision detection in Alice 3:
1. to use the in-built “collision detection” event listener feature, or
2. to manually specify threshold distance between 2+ objects

To get started

• Download the GhostFire_Start Alice 3 world.
• You will find that the objects are in place, with a ghost, four flames of different colors, and a coconut.
• Tip: If you want to change the color of an object, go to set up scene, click on the object, and on the right section, find its paint.
Guideline

• In Part I, we will use the in-built Alice 3 collision detector for ghost and coconut.
• In Part II, we will manually determine the threshold distance for collision between ghost and fire. Here, we will understand the underlying mechanism of the collision detector used in Part I.

Commanding the collision detector

• In the pop-up window, add “Coconut”. Click Ok.

• An identical window pops up. Add “Ghost.”

• In these two windows, you have specified which two objects the collision detector should keep an eye on. It should look like this:

```
declare procedure collisionStarted
  event getThingFromSetA
  event getThingFromSetB
```

pickUpCoconut Procedure

• When collision between coconut and ghost is detected, the ghost shall pick the coconut up.
• For clarity, let’s put this pickup in a scene procedure (as multiple parties are involved).
• Add Scene Procedure and name it pickUpCoconut
• You could improvise as you wish. A demo is displayed on the next page.
Finishing up Collision Detector

- First, switch back to the `InitializeEventListener` tab on the top.
- We have just written a procedure detailing what shall happen when the ghost and the coconut collide.
- We should now drag that procedure into the collision detector block, so as to execute the pickup procedure when collision happens.

Let’s enable the keyboard to control the ghost’s movement

*Add event listener >> keyboard>> add object move for >> ghost*

You can run the program and guide our floating buddy to the coconut! Remember: it moves from its own point of view.
Next Step – Ghost and Fire

• You have two options, either to add another collision detector for ghost and fire, or to manually set the threshold distance between the two beyond which you consider as collision.
• In Part II, we teach the latter method.

Part II: Manual adjustment of proximity/distance

• Click on the scene tab on the top bar and then find the properties section.
• Click on “Add Scene Property”. We will treat the four flames as a bloc.
• Is variable: constant field (because the four flames are all we need and unchanging)
• Value type: Tick “is array”. Then, click on Gallery Class – Flames. You’ll see that the four flames are automatically ticked on the left column.
• (continued on next page)

Adding the new property...

• Name: dangers
• Initializer: Click and select Custom Array, and then add all four flames.

Click ok when happy
What’s up next

• A ghost procedure called **jumpBack** as its reaction when colliding with any fire
• A function called **collide**, which
  • returns true if the ghost collides with ANY of the flames
  • returns false if the ghost collides with NONE of the flames
• A continuous listener of whether **collide** is true, and, if true, to execute **jumpBack**

Ghost procedure **jumpBack**

Click on the polygon and find ghost.
Create a new procedure and call it **jumpBack**. This will be executed when collision between ghost and fire occurs.

What’s up next

• A ghost procedure called **jumpBack** as its reaction when colliding with any fire
• A function called **collide**, which
  • returns true if the ghost collides with ANY of the flames
  • returns false if the ghost collides with NONE of the flames
• A continuous listener of whether **collide** is true, and, if true, to execute **jumpBack**
Scene function to determine collision

- Switch to the **scene** tab
- Add new scene function
  - Return type: boolean (true/false)
  - Name: collide
- Drag in a “for each in” from the bottom row
  - Item type: Gallery Class >> Flames
  - Item name: flame
  - Array: this.dangers (as we set up earlier)

Our goal is to turn the conditional to...

- **Goal:** Ghost distance to Flame < (ghost’s depth/2 + flame’s depth/2)
- Hence we replace the 1.0s on both sides respectively
  $1.0 < 1.0$
- Try figuring out on your own how to reach the goal, as the step by step process could look intimidating. Remember! Start with the exterior structure and work your way in like peeling an onion. Skip to page 31 to directly check your answer.

Scene function to determine collision

- Now, drag in an **if/else** statement within the previous block. Click on the “true” conditional and select **relational (decimal number)** <, and then any two placeholders such as 1.0 and 1.0.

Step by step

- **Goal:** Ghost distance to Flame < (ghost’s depth/2 + flame’s depth/2)
- **Step 1:** $1.0 < 1.0$
- **Step 2:** Ghost distance to Flame < (1.0 + 1.0) *shown on next two pages
Tip:
Click on ghost, and then the Functions tab. You’ll find useful elements like getDepth and getDistanceTo.

Step by step

• **Goal:** Ghost distance to Flame < (ghost’s depth/2 + flame’s depth/2)
• **Step 3:** Ghost distance to Flame < (1.0 / 2.0 + 1.0 / 2.0) *shown on next page

• Change both 1.0 to 1.0/2.0
Almost done

• **Goal:** Ghost distance to Flame < (ghost’s depth/2 + flame’s depth/2)
• **Step 4:** Ghost distance to Flame < (ghost’s depth/2.0 + flames4’s depth/ 2.0)
• Click on ghost and you’ll find getDepth in its functions. Click on any of the flames, say, flames4 (the red fire). Find getDepth and drag over the second 1.0. We use this flames4 as a placeholder.

The final step is the core. We want to check every flame’s collision status with the ghost, not just flames4. And so we need to replace the specific case of flames4 with the tracer `flame` created in the “for each in” loop to check every flame in the array dangers, which comprises all four flames.

The if conditional should look like this:

Fill out the function

• We need to return statements.
• The “return true” line goes inside of the if statement. We could return a positive answer for collision provided that any of flames collides with the ghost.
• The “return false” line goes outside of the entire for each in loop. We can only return the solid answer, that no, the ghost does not collide with any of the flames, after we have checked every single flame’s collision status with the ghost.
Almost done

- A ghost procedure called `jumpBack` as its reaction when colliding with any fire
- A function called `collide`, which
  - returns true if the ghost collides with ANY of the flames
  - returns false if the ghost collides with NONE of the flames
- A continuous listener of whether `collide` is true, and, if true, to execute `jumpBack`

Event listener

- Switch back to the **InitializeEventListeners** tab.
- We want the program to constantly check whether the ghost is in touch with any of the flames; and have it jump back if so.

Event listener

- Drag in a **while** from the bottom and select true.
- And then **within the while** loop, drag in an **if** block and select true.
Event listener

• Now, we want to constantly check collision status of ghost and fire.
• Find the “collide” function under “this”, and replace the “true” in the if block.

If collide, then jumpBack

Click on the ghost, find its jumpBack procedure and drag over under the if conditional. Every time that ghost collides with any of the flames, it will jump back.

Play it!

• I have a little fire dance that you could use if you want as a scene procedure. You could find it in ghostFire_Finished.
• Have fun!