4 Lab 4CS 123 Computation Lab III
Spring 2013
Directions and Problems

4.1 Overview

In this lab, we will create a simple game in Maple. A library API has been developed to assist in the creation of this game. In Part 0, we will examine the provided library. In Part 1, we will integrate this library into the Bouncing Ball Script from CS 122. In Part 2, we will use the Maple GUI components to create controls for the procedure we have developed.

This the way the game should work: We place a ball at (0,0) on the plot. We then fire this ball with a given velocity and angle. A collection of target blocks is also placed on the plot. We want the ball to hit the targets, causing them to disappear. For each box that disappears, we will score 1 point. To approximate the idea of gravity, any boxes directly above where the ball hits will also be destroyed.

In this image, the red line shows a possible path for the bouncing ball through the target.

---

<table>
<thead>
<tr>
<th>Example Showing Path of Ball</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score: 0</td>
</tr>
</tbody>
</table>

This second image shows the same path, but this time targets that have been hit have been removed and a score has been calculated.

| Path of Ball with Hit Targets Removed |
Directions and Problems

4.2 Pre-lab preparation

1. Read Chapter 19. Review older chapters and labs as needed. This lab uses the same simulation as CS 122 Lab 4, Part 2. You may want to review the lab directions from this lab. It will be useful to be familiar with the simulation used in that lab.

2. Review the CS123 Lab 3 GUI concepts, especially having sliders control plots. Make sure you understand the syntax to control GUI components.

3. Take the pre-lab quizlet 4 at the CS 123 Maple TA web site. You should do quizlet 4 before lab to be better prepared for the lab activities.

4. Read over the documentation for the new API provided in Part 0.

4.3 Problems

Part 0

In this Part, we want to examine the provided library. Understanding this library will be key to completing Part 1 of the lab. First, download the Lab 4 zip file from the course website. Open and execute the Lab4Part0.mw script. Read the description provided below, then complete the exercises in Lab4Part0.mw.

The game library consists of two files, bounceGame.ind and bounceGame.lib. These files can be included using the following maple commands.

```
libname := ".", libname;

#Load in the Bounce Game package
with(bounceGame);
```

Just as with CS122 car simulator, these commands will only work in the same directory as the library files. The command libname := ".", libname; is telling Maple where to look for the library files.

This library introduces 3 new procedures to Maple.
4.3 Problems

createTarget(x,y,w,h)

This procedure takes four inputs. The first two inputs are the (x,y) position of the bottom left corner of the target area. The second two parameters are the width and height of the target area. This is similar to the box proc created in Lab 3 in CS 122. The proc outputs a list of x,y points. Each of these points denotes the position of a 2 by 2 box in the target area.

The following command will create a 10 by 15 target area with the bottom left corner at (5,5).

T:=createTarget(5,5,10,15);

The output is a list of (x,y) pairs. For example, the following list might be the result of createTarget. The second element [5,7] tells us a box will be placed at position x=5, y=7.

[[5, 5], [5, 7], [5, 9], [5, 11], [5, 13], [5, 15], [5, 17], [5, 19], [7, 5], [7, 7], [7, 9], [7, 11], [7, 13], [7, 15], [7, 17], [7, 19], [9, 5], [9, 7], [9, 9], [9, 11], [9, 13], [9, 15], [9, 17], [9, 19], [11, 5], [11, 7], [11, 9], [11, 11], [11, 13], [11, 15], [11, 17], [11, 19], [13, 5], [13, 7], [13, 9], [13, 11], [13, 13], [13, 15], [13, 17], [13, 19]]

drawBoard(T)

This procedure takes a list of targets like the one created by createTarget. It converts the list into a plot structure that can be displayed in Maple.

Using a variable T created by createTarget, drawboard generates images like the following plot.

![drawBoard Example](image)

detectHits(x,y,radius,T)

The detectHits function performs collision detection. It takes 4 inputs. The first three inputs describe the ball the player is using in the game. The x and y inputs give the current center of the ball. The radius input gives the radius of the ball. The final input, Targets, is the current list of targets. The return value of this function is a new list of Targets. This new list is the old list with any targets the ball hit removed.

The example below updates the list of targets T with a ball hit at position x=12, y=10. The ball has a radius of 1. The result is then displayed using drawBoard.

T:=detectHits(12,10,1,T);

[[5, 5], [5, 7], [5, 9], [5, 11], [5, 13], [5, 15], [5, 17], [5, 19], [7, 5], [7, 7], [7, 9], [7, 11], [7, 13], [7, 15], [7, 17], [7, 19], [9, 5], [9, 7], [9, 9], [9, 11], [9, 13], [9, 15], [9, 17], [9, 19], [11, 5], [11, 7], [11, 9], [11, 11], [11, 13], [11, 15], [11, 17], [11, 19], [13, 5], [13, 7], [13, 9], [13, 11], [13, 13], [13, 15], [13, 17], [13, 19]]

drawBoard on T after detectHits
Directions and Problems

There are two additional Maple commands that you may find helpful.

\textbf{disk([x,y],radius,options)}

The disk command is part of the plottools library. We have already seen it in CS 121 and CS 122.

\texttt{p2:= disk([12,10],1,color="red");}

This command created a disk centered at (12,10) with radius 1 in the color red.

\textbf{sprintf(formatstring,inputs...)}

The sprintf command functions like the printf command. The only difference is, instead of printing the string immediately, it returns the string as its value so that the result can be stored into a variable for subsequent use.

For example, if you did

\texttt{plotname := sprintf("Score: %d",10);}

You could use the score in a plot tile: \texttt{plot(......,title=plotname,...).} This allows the plot tile to change according to the score when the code is executed.

Now that you understand the procedures required for this lab, let's do the exercises in the Lab4Part0.mw script.

\textbf{Part 1}

\textbf{(A) Adding the Library to the Bouncing Ball Script}

Open the Lab4Part1and2.mw script found in the downloaded zip file. Execute the script before beginning your work. Modify this script to complete the following tasks.

Using the tools from part 0, we want to add a target to this procedure. When the path of the ball hits a target, we can use the hit detection procedure to remove boxes.

You will need to make three major additions to the code. You may need additional changes to accomplish these goals.

1.) Create a set of targets before the "for" loop begins. Initialize a variable to track the score. The targets should be located at (50,0) with a width of 10 and a height of 10. Later in the lab we will change the position of the targets.

2.) Every iteration of the loop (at the start of the loop), remove any targets the ball has hit. Use detectHits to remove specific boxes in the target. Count the number of elements in the target list before and after the detectHits command. If the number of elements in the list changes, increase the score by the number of elements that have disappeared. (Hint: \texttt{nops(L)}return the number of elements in the list \texttt{L}.)

There are two additional Maple commands that you may find helpful.
3.) After the loop is complete, draw a final plot that shows only the targets missed. We want the plot to display the targets that were missed by the path. Add the score as the title of the plot. Use the display command to combine the target, plot, and title.

The return value of the function should look something like the below plot.

(B) Make the function return an Animation

At this point, the game mechanics are complete. To make the game more interesting to play, we will generate an animation instead of just the final plot.

You will need to make the following changes to the script.

1.) Create a frames table to store the animation before the start of the loop.

2.) During the loop (just after the score has been updated), add a new frame for the current animation.

frames[i] := A display of what the ball and boxes look like with the current score.

display([the ball, the drawBoard], title=the plot title containing the score)

3.) After the end of the loop, draw the animation rather than the plot. The score should be shown as the title of the animation and update as boxes are removed. Use the convert(...,list) and display(...,insequence=true) commands as we used in CS 122 Lab 4.

return display(convert(frames, list), insequence=true);
Part 2

Save a copy of your work for Part 1 as Lab4Part1and2_answer1.mw. Create a copy of of this worksheet and name it Lab4Part1and2_answer2A.mw. Use this version to complete Part 2 A.

(A) Creation of GUI interface

Instead of manually setting the angle and velocity then running the procedure, modify the worksheet to include GUI tools. Allow the user to set the velocity and angle, then fire the ball.

* Velocity should be a slider from 10 to 40.

* Angle should be a slider from 20 to 70.

Be sure to create the GUI on the same work sheet (and beneath) the code edit region containing your simulation.

Also, be sure to run the simulation prior to running the GUI for the 1st time.

**Game with GUI Components**
4.3 Problems

(B) Randomization of Target Position

Create a copy of this current worksheet and name it Lab4Part1and2_answer2B.mw. Use this version to complete Part 2 B.

Up to this point we have created a game with only one level. This may be fun initially, but once you have determined the best angle and velocity to hit the target, there is nothing else to do. The next step is level design. Most games use a combination of human design and computer randomization to keep levels interesting. For the final problem in this lab, we will introduce a small amount of randomization into the game we have created.

Maple provides multiple tools to generate random numbers. The RandomTools[Generate] function is one example.

To generate a number between 1 and 100 in maple, we would need to do the following.

First initialize the random number generator by executing the following command once at the beginning of the worksheet.

```
randomize();
```

Next, call the RandomTools[Generate] procedure to select a random number.

```
RandomTools[Generate](integer(range = 1 .. 100));
```

We can select any range we want for the numbers to be generated

```
RandomTools[Generate](integer(range = 7 .. 9));
```

Look up RandomTools[Generate] and randomize in the Maple Documentation to see more examples.

It is important to note the randomize() command. This command initialized the random number generator. It only needs to appear once before the first time you generate a random number. After randomize is called, you can make multiple calls to RandomTools[Generate].
1.) In your top code edit region (the one in which the libname is set), below the "with(plottools):" command create a new variable. You may pick a different range if you want. Remember to execute the code edit region after making the change.

   randomize();
   targetXPos := RandomTools[Generate](integer(range = 20 .. 60));

2.) Modify your procedure definition so that it takes a third parameter controlling the bottom left corner of the target area. Rename your procedure to bounceSimulation2.

   bounceSimulation2 := proc(v0, angle, targetXPos)
   You will need to modify your call to createTarget to use the new formal parameter (targetXPos) instead of a fixed value to define the lower left point of the target.

3.) Modify your calls to the bounceSimulation1 procedure to give an input for the new parameter. Remember to change the procedure call in your GUI button as well.

   bounceSimulation2(v0, angle, targetXPos);

Now, if you run the entire script again the boxes will be at a new location.

Save a copy of your work for Part 2 as Lab4Part1and2_answer2.mw

4.4 Final actions (end of class)

Email copies of your work to yourself and/or your partners. Be sure to get credit for doing this on the verification sheet before you leave.

4.5 Summary and conclusion

In this lab, we have used many of the concepts from previous labs in a new way. We have been introduced to new procedures that are specialized for our tasks. We have also used existing code to solve new problems. Many tasks in programming are better solved by using a combination of existing technologies. Every task could be started from scratch, but this is a waste if parts of the problem are already solved. By combining your knowledge of programming with code and libraries developed by others, you can create interesting programs.

Specifically, we used a combination of GUI components and functions provided by Maple and libraries created for this class to create a game. Without an understanding of the bouncing ball model from CS 122 and the GUI components from Lab 3 of CS 123, this task would have been much more difficult. By understanding the work of others and building on it, we have been able to create a more interesting and fun program.