Handout 5
CSCI 134: Fall, 2016

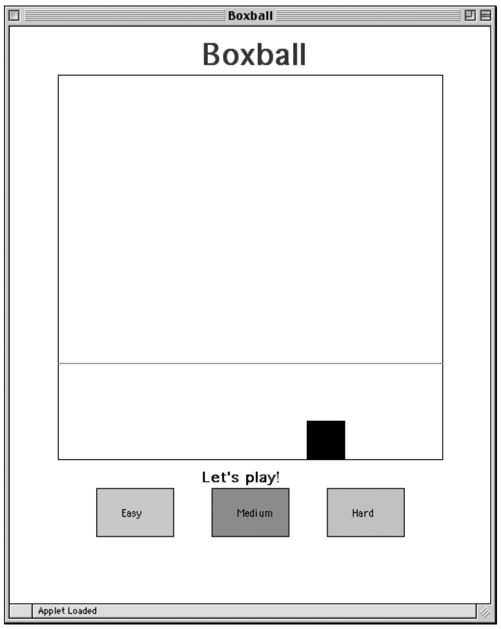
Lab 4

Boxball

Objective To gain experience defining constructor and method parameters and using active objects.

The Scenario This week you will implement Boxball, a simple game in which the player attempts to drop a ball into a box. Boxball has three levels of difficulty. With each increasing level, the box becomes smaller, and the player drops the ball from a greater height.

When the game begins, the playing area is displayed, along with three buttons that allow the player to select a level of difficulty. Selecting a level of difficulty changes the size of the box and the position of the line indicating the minimum height above which the ball must be dropped. The player drops a ball by clicking the mouse in the region of the playing area above the minimum height line. If the ball falls completely within the box, the box is moved to a new random location. Otherwise, it remains where it is. In either case, the player may go again, dropping another ball. The Boxball playing area should appear as follows:



The Handouts page on the cs134 website has a demo version of Boxball.

Design of the program

For this lab, you will define three classes: a `Box` class, a `Ball` class, and a `Boxball` class. The `Boxball` class extends `WindowController`. These classes allow you to create the major components of the game, i.e., the box, the balls, and the playing area, respectively.

You are expected to come to lab with a design for each class that corresponds to the functionality specified in parts 1, 2, and 3 described below. Clearly, you will probably want to further refine your initial design once you start writing the program, but keep in mind that you are responsible for writing up a *reasonable* approach to the whole program ahead of time. You may also wish to spend a bit of time in lab writing code to declare the necessary constants and set up the initial canvas layout prior to your lab session. This will enable you to focus on the more challenging aspects of Boxball during your lab session.

The `Boxball` class draws the playing area when the program starts. It also handles the player's mouse clicks. If the player clicks on the "Easy", "Medium", or "Hard" button, the starting line should move to the appropriate height. If the player clicks within the playing area above the starting line, a new ball should be created and dropped from that height.

The `Box` class allows you to create and manipulate the box at the bottom of the playing area. A box is responsible for knowing how to move to a new random location when the ball lands in the box. This class should also provide a method for changing the box size, so that the window controller can adjust it, based on the difficulty level.

The `Ball` class allows you to create a ball that falls at a constant rate. When the ball reaches the bottom of the playing area, the player should be told whether the ball landed in the box. The ball should then disappear. If the ball lands in the box, the box should move to a new location.

We provide starter code for these classes and include a copy of that code at the end of the handout for your reference.

Part 1: Setting up Start by setting up the layout of your playing area, using the familiar Object-Draw shapes. Our playing area (the framed rectangle in which the ball drops) has both a width and a height of 400. The canvas, however, needs to be larger to accommodate the buttons and title. The starter code we provide for the `Boxball` window controller class contains a call to the `resize` method in `begin`, which changes the size of the canvas to be 500 by 600 pixels.

Once you have set up the playing area, add the "Easy", "Medium", and "Hard" buttons to your layout. The buttons are just rectangles that will respond to mouse clicks. After you have displayed the three buttons, add code to the `onMouseClick` method to adjust the level of the starting line, depending on the button clicked. If the player selects the "Easy" buttton, the line should be relatively low. If the player selects the "Hard" button, the line should be quite high.

Part 2: Adding the box Next, you add a box to your layout. To do this, you will need to write the `Box` class that creates and displays the box object at the bottom of the playing area.

A `Box` object appears to be just a rectangle on the canvas, but there is some important information you will need to pass to the `Box` constructor in order to construct it properly. First, you need to tell the box where it should appear on the display. Remember that its horizontal position will change over time, but its vertical position will always be the same. What are the extreme left and right values for its horizontal position?

In order for the rectangle to be drawn, you will also need to tell the box what canvas it should be drawn on. This information will be passed on to the constructor for the rectangle.

The box needs one more piece of information for it to be drawn correctly, namely its width. Of course, even in the "Hard" setting, the box should still be a little wider than the ball. Since the `Boxball` class will create both the ball and the box, it knows their relative sizes. It must therefore tell the box how big it should be when the box is created.

Once you have written the constructor for the `Box` class, you should go back to the `begin` method of the `Boxball` controller class and create a new `Box`.

The default setting for the game is "Easy". If the player clicks on the "Medium" or "Hard" button, the box should get smaller (or much smaller). The box needs a method, `setSize`, to allow its size to change when the player clicks on "Easy", "Medium", or "Hard". Think carefully about what parameters you need to pass to `setSize` to accomplish this command.

After writing the `setSize` method, test it by modifying the `onMouseClick` method in the `Boxball` controller. Clicking on one of the level-selection buttons should now both reposition the bar and adjust the size of the box.

Part 3: Dropping a ball The `Ball` class is different from other classes with which we have been working. Objects of this class will be active.

The player creates a ball by clicking with the mouse in the playing area above the starting line. When the click is detected, a new `Ball` should be constructed. The constructor for the `Ball` class should draw a ball at the appropriate location on the screen. It should also start it moving down the screen. To do so, it will call a `start` method, which will cause the code in `run` to execute.

You will notice that the skeleton of the `Ball` constructor we have provided contains the call to `start`. You will need to add more statements to the constructor, but be sure that the `start` method call remains the last statement in your constructor.

Think carefully about what information a ball needs to know to construct itself properly. Recall that `Boxball` knows how big the ball should be (so that the ball and the box can be sized appropriately). `Boxball` also knows where the mouse was clicked. This should be the starting location for the ball. You need to pass this information to the `Ball` constructor so that it can draw itself and fall.

To make the ball fall, you add code to the `Ball`'s `run` method. We have provided a skeletal `run` method for you.

We will use a bit of video game magic to make the ball appear to fall. The ball appears to move smoothly down the screen, but in fact, it moves by a series of discrete movements. Each movement should move the ball a short distance, wait a short time, and then move again. The `run` method contains a while loop to allow this. On each iteration of the while loop, the ball should move a small distance, say 5 units. Then it should wait a short time. The pause method call that we provided in the starter tells the ball to wait 30 milliseconds. There are 1000 milliseconds in a second. Moving short distances that rapidly will appear to be continuous movement to the human eye. This is the same technique that television and movies use to provide continuous motion. To complete the while statement, you should provide the condition that determines when to exit the while loop. Specifically, the ball should stop moving when it reaches the bottom of the playing area.

Once you have written the `Ball` constructor and the `run` method, test them. Return to your `Boxball` controller class, and add code to the `onMouseClick` method to construct a ball if the player clicks above the starting line in the playing area. At this point, do not worry about whether the ball falls in the box. Just check that it is drawn at the right starting location and that it makes its way to the bottom of the playing area.

Part 4: Checking the box Now you are ready to determine whether the ball fell in the box. As the ball reaches the bottom of the playing area, it should compare its location to the box's location. Of course, the ball will need to find out the box's location. The `Box` class needs to provide methods `getLeft` and `getRight` that give the positions of the edges of the box. To call those methods, the `Ball` class must know about the box. Go back and modify your `Ball` constructor to pass in the `Box` as an additional parameter.

If the ball lands in the box, display the message "You got it in!". If the ball misses, display "Try again!". Since the `Boxball` controller is responsible for the layout of the game, it should construct a `Text` object that displays a greeting message. The `Text` object should be passed to the `Ball` constructor as a parameter so that the ball can change the message appropriately when it hits or misses the box.

Test the additions that take care of checking whether the ball fell in the box.

Finally, use a random number generator to pick a new location for the box when the player gets the ball in the box. The box should be responsible for picking the new location and moving itself. Therefore, you will need to add a method to the `Box` class called `moveBox`.

When the box is narrow its left edge can have a relatively large value while still having the whole box in the playing area. However, when the box is wide, it cannot move quite as far without having its right edge going outside the playing area. You may set up the random number generator to only give values that will result in the widest box staying inside the playing area. For extra credit, you can further refine the program so that even the narrower boxes can end up all the way on the right of the playing area.

Getting Started

We have provided a starter project for you to use. To download the starter project, visit

```
http://www.cs.williams.edu/~cs134/
```

and then follow the link to the Handouts page. Click on the link for "Starter Code" under Lab 4.

This will download a file archive called `Lab4Boxball.tar.gz` to your cs134 folder. The files of this archive will then be extracted to a folder in your cs134 folder called `Lab4Boxball`. (If this does not happen automatically, simply double-click on the downloaded `Lab4Boxball.tar.gz` file to extract the archive.) You may delete the `Lab4Boxball.tar.gz` file at this point. Rename the "Lab4Boxball" folder to include your last name, as usual, and open the project folder in BlueJ.

Submitting Your Work

Once you have saved your work in BlueJ, please perform the following steps to submit your assignment:

- First, return to the Finder. You can do this by clicking on the smiling Macintosh icon in your dock.
- From the "Go" menu at the top of the screen, select "Connect to Server...".
- For the server address, type in "afp://Guest@fuji" and click "Connect".
- A selection box should appear. Select "Courses" and click "Ok".
- You should now see a Finder window with a "cs134" folder. Open this folder.
- You should now see the drop-off folders for the three lab sections. Drag your "Lab4Boxball" folder into the appropriate lab section folder. When you do this, the Mac will warn you that you will not be able to look at this folder. That is fine. Just click "OK".
- Log off of the computer before you leave.

You can submit your work up to 11 p.m. on Wednesday if you're in the Monday night lab; up to 6 p.m. on Thursday if you're in the Tuesday morning lab; and up to 11 p.m. on Thursday if you're in the Tuesday afternoon lab. If you submit and later discover that your submission was flawed, you can submit again. We will grade the latest submission made before your lab deadline. The Mac will not let you submit again unless you change the name of your folder slightly. It does this to prevent another student from accidentally overwriting one of your submissions. Just add something to the folder name (like the word "revised") and the re-submission will work fine.

Grading Guidelines

As on previous labs, we will evaluate your program for both style and correctness. Here are some specific items to keep in mind and focus on while writing your program:

Pre-Lab Design

- Boxball class
- Box class
- Ball class

Style

- Descriptive comments
- Good names
- Good use of constants
- Appropriate formatting

- Good use of boolean expressions
- Not doing more work than necessary
- Using most appropriate methods
- Good use of if and while statements
- Good choice of parameters

Correctness

- Drawing the game correctly at startup
- Changing box size and line height correctly
- Dropping the ball
- Determining if the ball landed in the box
- Moving the box after the ball lands in it

Starter Code

Boxball Class

```
import objectdraw.*;
import java.awt.*;

public class Boxball extends WindowController {

    private static final int WINDOW_WIDTH = 500;
    private static final int WINDOW_HEIGHT = 644;

    public void begin() {
        // Resize the window to be 500 x 644 pixels in size.
        // Since the window's menubar is 44 pixels high,
        //  this call makes your canvas be 500 x 600 pixels.
        this.resize(WINDOW_WIDTH, WINDOW_HEIGHT);
    }

    public void onMouseClick(Location point) {   }
}
```

Box Class

```
import objectdraw.*;
import java.awt.*;

public class Box {

    // You need to add the parameters.
    public Box() {   }

    // Move the box to a random location at the bottom of the playing area.
    // Add parameters if necessary.
    public void moveBox() {    }

    // Set the width of the box.
    // Add parameters if necessary.
    public void setSize() {    }

    // Return the x position of the left edge of the box.
    // Replace the return statement with the correct code.
    // Add parameters if necessary.
    public double getLeft() {
        return 0.0;
    }

    // Return the x position of the right edge of the box.
    // Replace the return statement with the correct code.
    // Add parameters if necessary.
    public double getRight() {
        return 0.0;
    }
}
```

Ball Class

```
import objectdraw.*;
import java.awt.*;

public class Ball extends ActiveObject {

    private static final int PAUSE_TIME = 30;

    // Add the parameters that you need.
    public Ball() {
        // Add your code here.

        // Make sure that the start method call is the last statement
        // in the constructor.
        this.start();
    }

    public void run() {
        // Change the true condition to the condition that tests for
        // the ball reaching the bottom of the playing area.
        while (true) {
            // Add code to move the ball a small distance down here.
            pause(PAUSE_TIME);
        }
        // Add the code that should be executed when you reach
        // the bottom of the playing area.
    }
}
```