CIS 201 – Computer Science I
Laboratory Assignment 5

Introduction

In this lab you will create a game that will incorporate

- Count controlled iteration (`for` statement)
- Non-text score keeping

The purpose of this game is to use the arrow keys to move a "rake" around the active part of the game canvas to pick up leaves and to deposit them in a separate leaf pile.

Raking Leaves

Create a `Lab05` in which you will do your work, and change to that directory. Begin editing a game with the class name `RakingLeaves` as outlined below. Remember that this game will extend the FANG `Game` class and that you will need to use an `import` statement for this to work. Include a header comment, a class declaration, and our standard two methods. Here is your starting point.

```
import fang.core.Game;

/**
 * Describe class RakingLeaves here.
 *
 *
 * Created: Tue Sep  2 09:37:57 2008
 *
 * @author <a href="mailto:student1@potsdam.edu">Indiana Jones</a>
 * @author <a href="mailto:student2@potsdam.edu">Luke Skywalker</a>
 * @version 1.0
 */
public class RakingLeaves extends Game {

  /**
   * Deposit the leaves into the raking area and set up the rake
   */
  public void setup() {
```

```
  }

  /**
   * Move the rake to pick up the leaves, and deposite them
   * into the leaf pile.
   *
   * @param dT time (in fractional seconds) since the last call to advance
   */
  public void advance(double dT) {

  }
} // class RakingLeaves
```

Change the size of the game. We want to give up some of the screen to a "scoring box" so you will want to make the game field larger than it normally is. Remember that though we change the size (and shape) of the screen, location coordinates are still given in *screens* so the upper-left corner is still at `(0.0, 0.0)` and the lower-right corner is still at `(1.0, 1.0)`.

Game size is set during construction of the game. You must add a *constructor* to your game. A constructor is a special method that is called when an object of the given type is created (when `new` is called). Your game extends the FANG `Game` class, and the FANG game engine automatically constructs an instance of your game to get things going. If you don't provide your own constructor, the FANG library gives you one free of charge – the one you have already seen on your earlier games.

The signature of a constructor has no return type and has the same name as the class itself (two ways to recognize that a method is a constructor if you were ever asked on a test or quiz, for example). So, a `public` constructor for our `RakingLeaves` class with an empty parameter list would look like this:

```
public RakingLeaves () {
...
}
```

In a constructor, the very first line can call the superclass constructor (in this case, the constructor for the FANG `Game` class), possibly providing it with different parameters. The way you specify the "other" constructor is with the keyword `super`. This must be the first non-comment line of the constructor. In our case, it will be the only line: we will call the `Game` constructor that takes a width and height for the game field *in pixels*. Your game should be 640 by 640 pixels (width by height), so you should put this constructor in your code, just after your `class` declaration:

```
public RakingLeaves() {
```

```
    super(640, 640);
  }
```

This constructor requires a header comment just like we have for our methods. Add an appropriate comment to your code.

Checkpoint 1

Compile and run your program. Show us your code and that your game canvas appears to be larger on the screen than before.

Setting up the game

Leaf Pile

Create a "leaf pile": a green rectangle filling approximately the left 1/3 of the screen. This is where you will pile your leaves. Declare a double field variable named `split` that will be the *x*-coordinate at which you will split your screen. A value of 0.33 will be fine, but you can change this later if you wish.

To create the leaf pile, define a `RectangleSprite` variable in your `setup` method, giving the rectangle the appropriate dimensions (width `split` and height 1.0) and color (green) and adding it to your game canvas. You should *not* declare the variable as a field variable since you will never need to refer to it outside of the `setup` method.

Making leaves

Define a `makeLeaf` method. This method will create a new leaf and return it to the caller. Putting this task in its own method will make it easy to reuse it when we need to update the score and add new leaves to the game.

The method should return a `PolygonSprite` because our leaves will be regular polygons with 3-6 sides. The method should declare a local variable to hold the leaf (since you will need to set color, scale, etc. before returning it).

The following code should start you out This code defines a variable `leaf` to hold a `PolygonSprite` and returns the `leaf` to the caller. Put this code *after* the `setup` and `advance` methods.

```
public PolygonSprite makeLeaf() {
  PolygonSprite leaf = new PolygonSprite(...); // fill in...
  // <fill in according to discussion below>
  leaf.setLocation(/* see below */);
  return leaf;
}
```

If you were to try to compile this code, it would give you errors, particularly because of the three dots! You need to replace it with an expression that returns a random integer value between 3 and 6 (inclusive). You should use the `randomInt` method, part of the `Game` class, to get such an integer. Read the documentation to find out what parameters this method requires.

After you have created the leaf, you will need to set its color randomly. You can use the `getColor` method to set the color appropriately, using random values for red, green, and blue. But don't make the colors too dark, otherwise you won't be able to see them on the canvas. Thus you should use random values between 128 and 255 for each of the red, green, and blue values. Set the scale of the leaf to 0.05 using the `setScale` method.

We want to randomly position the leaf to the right of the leaf pile. Use the `randomDouble` method (see the documentation) to position the leaf to the right of the leaf pile. Remember that the `split` variable gives the *x*-coordinate where the leaf pile ends and the rest of the game canvas (the yard) starts. Use this code:

```
leaf.setLocation(randomDouble(0.33, 1.0),randomDouble());
```

Finally, add the leaf to the game canvas.

Do all of this *before* you return the `leaf` to the caller.

Add leaves to the yard

In your `setup` method, You will add 10 leaves to the yard (to the right of the leaf pile). You will need to keep track of each leaf, but you don't want to have a separate variable for each. To do this, you declare a field variable `leaves` which will be an `ArrayList` of `PolygonSprite`s. Here is the declaration:

```
private ArrayList<PolygonSprite> leaves;
```

You will need to use an import statement to load the `ArrayList` class from the Java class library:

```
import java.util.ArrayList;
```

Near the beginning in your `setup` method, define your `leaves` variable as follows:

```
leaves = new ArrayList<PolygonSprite>();
```

The `<PolygonSprite>` part tells Java that the array list can only hold `PolygonSprite`s, and nothing else.

After you have defined the `leaves` variable in the `setup` method, you can create 10 leaves, keep track of them in the `leaves` array, and add them to the game canvas. Here is the code:

```
for (int i = 0; i < 10; i++) {
  PolygonSprite nextLeaf = makeLeaf(); // why we made a method
  leaves.add(nextLeaf);
}
```

Checkpoint 2

Show us your work at this point. When you run your program, you should see a bunch of leaves randomly placed in the "yard".

Raking leaves

Make a rake

A rake is a yellow triangle with a scale of 0.10. Initially position it in the middle of the screen. The `rake` variable will be a `PolygonSprite` with three sides. Declare it as a field variable and initialize it in the `setup` method. Be sure to add it to your game canvas!

Moving the rake

In the `advance` method, use the arrow keys to move the rake. The rake should move at the rate of about 2 screens/second. Create a field variable defining the rake speed and use it in the `advance` method. Be sure that the rake doesn't leave the yard. This means that you need to should *not* move the rake horizontally unless its *x* coordinate is between `split` and 1.0, and vertically unless its *y* coordinate is between 0.0 and 1.0.

Checkpoint 3

Show us how your rake moves.

Piling up leaves

When the rake “rakes over” a leaf, that leaf should be put into the scoring box (position to be described below) and a new leaf is to be added randomly to the yard.

To check if a the rake hits a leaf, you should use the `intersects` method. Since all of the leaves in the yard are stored in the `leaves` array, all you need to do is to march through the array and check for intersections. Here is the code to do so:

```
for (int i=0 ; i<leaves.size() ; i++) {
  PolygonSprite leaf = leaves.get(i);
  if (rake.intersects(leaf)) {
    // move the leaf to a random place in the bottom half
    // of the leaf pile
    leaves.set(i, makeLeaf());
  }
}
```

Checkpoint 4

Show us your work.