CMSC110 Introduction to Computing

Lab #10: Ghosts! (Getting to learn OOP)
Week of November 7, 2016

Let us design an object, say a Pacman ghost, and how to model it using OOP design techniques we are learning in class. Look at the following description:

> A *ghost* is a shape (as shown on the right) that can be *drawn* in a sketch. In the Pacman game there are four ghosts identified by their colors: Blinky (red), Clyde (yellow), Pinky (pink), and Inky (light blue). It can *move* from right to left (or from left to right) in a sketch. When it goes out of the sketch boundary it reappears on the other side.

Note the italicized words in the description. For us a ghost is an object: a shape (with x- and y-coordinates, a color, and a name). Additionally a ghost can be drawn and moved about in a sketch. Here is a design of a ghost class:

OOP Vocabulary

Object
Class
Instance
Attributes
Constructors
Methods
Accessors
Modifiers
Print Methods

Class Ghost
Attributes: position (**gx**, **gy**), color (**gcolor**), and a name (**gname**).
Behaviors: draw a ghost (**display()**), and move it (**move()**)

From the above, we can see that we will need 4 attributes (**gx**, **gy**, **gcolor**, and **gname**) for a ghost. We will also need **display()** and **move()** methods to enable drawing and moving. Additionally, every class definition provides one or more constructors that enable a user to create instances of ghosts.

In our initial sketch, our plan is to create a single ghost object and have it move across the screen from left to right. It should reappear on the left after it disappears on the right.

Later, we will add the capability of other ghosts as well as some interactivity. We will implement our design incrementally.

Below, we define a skeletal class for modeling ghosts. Create a new tab in your Processing window named Ghost. Then add this code:

```
class Ghost {

  float gx, gy;       // location (bottom-left corner)
  color gcolor;       // color
  String gname;       // name

  final color EYE_COLOR = color(17, 100, 131);
  final float SIZE = 80; // size is 80 pixels

  Ghost() {         // Default constructor
    gx = 10;         // located at <10, height/2>
    gy = height/2;
    gcolor = color(255, 0, 0);  // red - Blinky!
    gname = "Blinky";
  } // Ghost
```

```
  void display() {
    final float dx = SIZE/6;
    final float dy = SIZE/5;
    final float yDisp = -2*SIZE/3;

    pushMatrix();
    translate(gx, gy);
    fill(gcolor);    // Draw body in gc color
    noStroke();
    beginShape();
    vertex(0, 0);
    vertex(0, yDisp);
    vertex(SIZE, yDisp);
    vertex(SIZE, 0);
    vertex(SIZE-dx, -dy);
    vertex(SIZE-2*dx, 0);
    vertex(SIZE-3*dx, -dy);
    vertex(SIZE-4*dx, 0);
    vertex(SIZE-5*dx, -dy);
    endShape(CLOSE);

    arc(0.5*SIZE, 0.9*(yDisp), SIZE, SIZE, PI, TWO_PI);  // Draw head

    float eyeS = SIZE/4;  // eyes
    fill(255);
    ellipse(SIZE/2-eyeS/2-5, yDisp, eyeS, eyeS*1.2);
    fill(EYE_COLOR);
    ellipse(SIZE/2-eyeS/2-9, yDisp, 1.2*eyeS/2, 1.2*eyeS/2);
    fill(255);
    ellipse(SIZE/2+eyeS/2+5, yDisp, eyeS, eyeS*1.2);
    fill(EYE_COLOR);
    ellipse(SIZE/2+eyeS/2+1, yDisp, 1.2*eyeS/2, 1.2*eyeS/2);
    popMatrix();
  } // display()

  void move() {
  } // move()
} // class Ghost
```

Before implementing this in your own Processing sketch, please study the class, and its structure carefully. If you remove the details from the **display()** method (you may take those for granted for now as all they do is issue commands to draw the design of the ghost shape). Notice that we added constants to the definition of the **Ghost** class to specify aspects of a ghost that do not change. Here is the structure of the class with details removed:

```
class Ghost {

   // Define attributes below
  float gx, gy;      // location (bottom-left corner)
```

```
  color gcolor;      // color
  String gname;      // name

  final color EYE_COLOR = color(17, 100, 131);
  final float SIZE = 80; // size is 80 pixels

  Ghost() {        // Default constructor
     // for each instance, set up its attributes
  } // Ghost

  void display() {
      // when this is called on an object, the object is drawn
  } // display()

  void move() {
      // when this is called on an object, the object's location is
      // updated
  } // move()
} // class Ghost
```

In order to complete the first version of the sketch (it will create one instance of a ghost and draw it on the screen), you will need the following main program:

```
Ghost blinky;   // the ghost object

void setup() {
  size(800, 400);
  blinky = new Ghost();
} // setup()

void draw() {
  background(0);
  blinky.display();
} // draw()
```

When you run the sketch above, you will see **Blinky**, the red ghost displayed in the sketch.

Moving Blinky

Next, extend the sketch to move **Blinky** from left to right in the sketch. First, define a new attribute, **`deltaX`** in the **Ghost** class:

```
int deltaX = 1;  // rate at which ghost moves
```

This variable represents how much, in the *x*-direction, the ghost moves. Next, change the **`move()`** method:

```
void move() {   // Move the ghost by deltaX
    gx += deltaX;
    if (gx > width) { // if it goes off the right edge, it starts back
at the left edge
```

```
      gx = -SIZE;
    }
} // move()
```

Finally, add the command:

```
blinky.move();
```

To the **draw()** method of your sketch (just before the call to **display()**). Run the sketch to see the outcome. Blinky should travel all the way across the sketch, past its right edge, and reappear on the left edge.

Interactive Ghosts

Let us implement some simple interactivity with the one ghost object we have in our sketch. Let's say that if we click the mouse button on the ghost, it will respond with some behavior. For starters, so that we can keep this simple, the behavior would be for the ghost to just print its name in the console window. Later, we will enhance this.

In your main program, add the **mouseClicked()** method shown below:

```
void mouseClicked() {
  blinky.clicked(mouseX, mouseY);
} // mouseClicked()
```

It is very simple: whenever the mouse is clicked, it may or may not be on the ghost, but for now it just calls a method called **clicked()** on object (Blinky). The **clicked()** method needs to be defined in the **Ghost** class:

```
void clicked(float mx, float my) {
    // mouse clicked on me, print out my name in console
    if (mx >= gx && mx <= (gx+SIZE) && my < gy && my >= (gy-SIZE)) {
      println(gname);
    }
  } // clicked()
```

The **clicked()** method is sent the *x*- and *y*-coordinates of the mouse. It then checks to see if the mouse was clicked in the rectangle that encloses the ghost shape. If so, it prints out the name of the ghost. Otherwise, it ignores it.

Go ahead, and place the two functions above as described and run your sketch. Try to click on the ghost as it moves, and observe that its name appears in the console window.

Customized Ghosts: What about Inky, Pinky, and Clyde?

Next, we can extend the program to create other, customized ghosts. Instances of **Ghost** can be created by calling the constructors. So far, our constructor creates only the red ghost (Blinky). Let us define another constructor:

```
  Ghost(float x, float y, color c, String name) {
    gx = x;         // located at <x, y>
    gy = y;
    gcolor = c;
    gname = name;
  } // Ghost
```

Next, update your program to the following:

```
Ghost inky, pinky, blinky, clyde;

void setup() {
  size(800, 400);
  blinky = new Ghost(random(width), random(90, height-10),
                     color(255, 0, 0), "Blinky");
  inky = new Ghost(random(width), random(90, height-10),
                   color(78, 206, 222), "Inky");
} // setup()

void draw() {
  background(0);
  blinky.move();
  blinky.display();
  inky.move();
  inky.display();
} // draw()

void mouseClicked() {
  blinky.clicked(mouseX, mouseY);
  inky.clicked(mouseX, mouseY);
} // mouseClicked()
```

Now, run the sketch. Click on the two ghosts and observe the behavior. Do you see the names of the ghosts printed out?

Go ahead and create Pinky (**`color(250, 136, 138)`**), and Clyde (**`color(247, 200, 15)`**).

Now that we are passing parameters to the constructor, we can remove the original, 0-parameter Blinky constructor from **`Ghost`.**

Making it more interesting: (1) Movement

Next, in order to make this sketch more interesting, let us focus on how the ghosts move. First, Modify the **`Ghost`** constructors so that the amount they move is a random value between -2 and 2. That is, add the command:

```
deltaX = int(random(-2, 3));
```

to the constructor. Additionally, now that ghosts may also move from right to left, you will need to add the following check in move():

```
if (gx+SIZE < 0) {
        gx = width;
}
```

That is, if the ghost leaves the left edge of the screen, it reappears on the right edge. Try it!

Making it more interesting: (2) Many, many ghosts!

As you can see, as the number of ghosts increases, the number of variables, and also the commands on them increases, and becomes tedious and repetitive. To resolve this, we can have an array of ghosts!

So that we can still assign proper names and colors, we will set up the following arrays in our main sketch. Now, the rest of the program can be written as shown below:

```
final String[] NAMES = {"Blinky", "Inky", "Clyde", "Pinky"};

final color[] COLORS = {color(255, 0, 0), color(78, 206, 222),
                        color(247, 200, 15), color(250, 136, 138)};
Ghost[] ghosts = new Ghost[4];

void setup() {
  size(800, 400);
  for (int i=0; i < ghosts.length; i++) {
    ghosts[i] = new Ghost(random(width), random(90, height-10),
COLORS[i], NAMES[i]);
  }
} // setup()

void draw() {
  background(0);

  for (int i=0; i < ghosts.length; i++) {
    ghosts[i].move();
    ghosts[i].display();
  }
} // draw()

void mouseClicked() {
  for (int i=0; i < ghosts.length; i++) {
    ghosts[i].clicked(mouseX, mouseY);
  }
} // mouseClicked()
```

Functionally, this program is essentially the same as before, but is now more flexible and extendible! You can now add many, many ghosts! Spooky!!

Completing the Ghost class: Accessors and Modifiers

It is time now to complete the **`Ghost`** class so that it includes accessor and modifier methods. Add these to the **`Ghost`** class:

```
  String getName() {
    return gname;
  } // getName()

  void setName(String name) {
    gname = name;
  } // setName()

  color getColor() {
    return gcolor;
  } // getColor();

  void setColor(color c) {
    gcolor = c;
  } // setColor()

  void switchDirection() {
    deltaX = -deltaX;
  } // switchDirection()
```

Even More Interesting: (3) Interaction

Suppose you click on a ghost and it responds by switching itself to be another ghost! Let us try this:

First, change the **`clicked()`** method to the following:

```
  boolean clicked(float mx, float my) {
   // mouse clicked on me?
    if (mx >= gx && mx <= (gx+SIZE) && my < gy && my >= (gy-SIZE)) {
      return true;
    }
    return false;
  } // clicked()
```

A shorter way of writing this function above is:

```
boolean clicked(float mx, float my) {
   // mouse clicked on me?
    return mx >= gx && mx <= (gx+SIZE) && my < gy && my >= (gy-SIZE);
  } // clicked()
```

Why does this work?

Next, we will use the accessor and modifier methods to do a switch whenever a ghost is clicked. Replace the **`mouseClicked()`** method by the one below:

```
void mouseClicked() {
  for (int i=0; i < ghosts.length; i++) {
    if (ghosts[i].clicked(mouseX, mouseY)) {
      int j = int(random(0, ghosts.length));  // pick a random ghost,
j
```

```
      if (i != j) {
        println(ghosts[i].getName()+" is switching identity with
"+ghosts[j].getName());
        color tempColor = ghosts[i].getColor();
        String tempName = ghosts[i].getName();
        ghosts[i].setName(ghosts[j].getName());
        ghosts[j].setName(tempName);
        ghosts[i].setColor(ghosts[j].getColor());
        ghosts[j].setColor(tempColor);
        ghosts[i].switchDirection();
      }
    }
  }
} // mouseClicked()
```

Your Turn

Switching Identity: Run the program several times. Note how the color and names of the ghosts are swapped in **`mouseClicked()`** using the temporary variables. Write a method **`switchIdentity()`** that can be used to swap the color, name, as well as direction using a single call:

```
ghosts[i].switchIdentity(ghosts[j]);
```

Collisions: Further enhancements are possible. One you may think about is to detect a collision between two ghosts. If two ghosts collide, they can switch identities. The Processing function **`dist()`** can be used to detect collisions:

`dist(x1, y1, x2, y2)` returns the distance between any two points <x1, y1> and <x2, y2>. You can use the center points of the ghosts to detect collisions.

Passive Ghosts: Say, after some time (you determine) the ghost goes passive (turns grey and cannot be clicked) for a little while (you determine).