Drawing and Controling the World

Viera K. Proulx

February 8, 2005

Simple Drawings

In order to draw the shapes like the circles, rectangles, and the combo shapes we have seen, we must have some place where the drawings will become visible. Doing so requires that we use a tool (teachpack) that provides the drawing surface and allows us to control its contents. We will first learn to draw basic shapes. We will then see how the drawing can change over the time (creating an animation), or in response to some events, such as a key press. These three features: the drawing canvas, the change over time, and the response to keyboard, mouse, or other events is at the core of many computer applications today.

To be able to see the drawing there must be a world (a stage, a canvas) where the drawing becomes visible. *ProfessorJ* provides such world through a teachpack and a library. Add the teachpack htdp/draw.ss to your environment, and start the code with the follwing import statements:

```
import draw.Posn;
import draw.World;

import draw.Color;
import draw.Red;
import draw.White;
import draw.Green;
import draw.Blue;
```

This is the way we can use classes that have been defined elsewhere. The last five import statements provide classes that represent colors for our drawings. So, if we want our circle to be red, we must specify the red color as

```
new Red()
```

The first statement imports the `Posn` class, thus liberating us from defining it every time we want to deal with drawings.

The statement `import draw.World` provides for us the class `World`. The `World` contains the following useful methods:

```
// draw a solid disk
boolean drawDisk(Posn center, int radius, Color c);

// draw a solid rectangle
boolean drawRect(Posn nw, int width, int height, Color c);

// draw a circle
boolean drawCircle(Posn center, int radius, Color c);

// draw a line
boolean drawLine(Posn from, Posn to, Color c);

// clear a solid disk
boolean clearDisk(Posn center, int radius, Color c);

// clear a solid rectangle
boolean clearRect(Posn nw, int width, int height, Color c);

// clear a circle
boolean clearCircle(Posn center, int radius, Color c);

// clear a line
boolean clearLine(Posn from, Posn to, Color c);

// create a drawing canvas of the given size and show it
boolean start(int width, int height);

// destroy the drawing canvas
boolean stop();
```

We now design a simple class that represents a red blob on the canvas, in the shape of a circle. We include a method `draw` that will draw the blob in the given world. We add two other methods as well. One method that moves the blob by a given distance, and, of course, the method `same` that compares two blobs for extensional equality, so that we can test the method `move`.

```
// to represent a blob on the canvas
class Blob {
  Posn center;
  int radius;

  Blob(Posn center, int radius) {
    this.center = center;
    this.radius = radius;
  }
  // draw this blob in the given World
  boolean draw(World w) {
    return w.drawDisk(this.center, this.radius, new Red());
  }

  // produce a circle with the center moved by the given distance;
  Blob move(int dx, int dy){
    return new Blob(new Posn(this.center.x + dx,
                             this.center.y + dy),
                    this.radius);
  }

  // determine whether this blob is the same as than given blob
  boolean same(Blob b){
    return this.center.x == b.center.x
        && this.center.y == b.center.y
        && this.radius == b.radius;
  }
}
```

Of course, we make examples beforehand. You have already made similar examples in your earlier homeworks.

```
class Examples {
  Examples() {}

  // Examples of blobs: c1, c2, c3 ---------------------------------
  Blob b1 = new Blob(new Posn(100, 300), 30);  // center low
  Blob b2 = new Blob(new Posn(180, 20), 20);   // top right corner
  Blob b3 = new Blob(new Posn(40, 360), 40);   // bottom left corner

  // Testing the same method in the class Circle --------------------
  boolean testSame1 = b1.same(new Blob(new Posn(100, 300), 30));
  boolean testSame2 = b1.same(new Blob(new Posn(100, 300), 50)) == false;
  boolean testSame3 = b1.same(new Blob(new Posn(130, 300), 30)) == false;
  boolean testSame4 = b1.same(new Blob(new Posn(100, 140), 30)) == false;


  // Testing the move method in the class Circle -----------------
  Blob movedcircle = b1.move(50, 50);
  boolean testMove1 = this.movedcircle.same(
                          new Blob(new Posn(150, 350), 30));
  boolean testMove2 = this.b2.move(-20, 20).same(
                          new Blob(new Posn(160, 40), 20));

}
```

When creating and moving drawings, we can (and will) still test, whether the new *moved* object has been constructed as expected.

To actually display the `Blob` on a canvas, we need an instance of the `class World`, tell the world to produce a canvas of a given size, and then draw the shapes.

The following code in the `class Examples` does the job:

```
// Building the drawing world ---------------------

// make an example of a drawing world
World dw = new World();

// show the world as a canvas of the size 200, 400
boolean testWorldStart = this.dw.start(200, 400);

// Drawing the circles:" ------------------------------------
// make a big blob and a small blob
Blob bigBlob = new Blob(new Posn(100, 200), 50);
Blob smallBlob = new Blob(new Posn(100, 100), 25);

// draw the big blob and small blob in the given world
boolean testBigBlobDraw = bigBlob.draw(this.dw);
boolean testSmallBlobDraw = smallBlob.draw(this.dw);
```

We first make an instance of the `World`, then tell the world to create a canvas of width 200 and height 400, and make it visible, by invoking its `start` method. Once we have instances of `Blobs`, they can invoke their `draw` method using the `World` we built as their argument. If we include this code in the class `Examples`, the instantiation of the class will produce the canvas and the drawing.

Event-Driven World

Next we learn to write programs that respond to user-initiated events, such as keystrokes. Our goal is to move the blob around the canvas in response to the four arrow keys: *up, down, left, right.* At each such keystroke we move the blob a fixed distance in the indicated direction.

To accomplish this, we need to add to the class Blob the method

```
  // move this circle 20 pixels in the direction given by the ke
  Blob moveBlob(String ke){ ...
```

Exercise: Follow the design recipe to develop this method. Test it properly.

We then define a new class KeyEventWorld that extends World. This class has one field that represents the blob to be drawn:

```
// represent the world of a Blob
class KeyEventWorld extends World {
   Blob blob;
   ...
```

We first add the method that will draw the blob in this world:

```
   // draw this world
   boolean draw() {
     return this.blob.draw(this);
   }
```

Next, we need to add to our KeyWorld class the method that will invoke the move method in the Blob class in response to each keystroke, with the String argument that represents that keystroke. The method is defined as follows:

```
   // what happens when the player presses a key
   World onKeyEvent(String ke) {

     // return the new world with the moved blob
     return new KeyEventWorld(this.blob.moveBlob(ke));
   }
```

We now construct an instance of the KeyEventWorld and start the world's timer. In the class Examples we need the following:

```
  // construct an instance of a KeyEventWorld
  KeyEventWorld w = new KeyEventWorld(new Blob(new Posn(100, 200), 20));

  // start the world, start the timer
  boolean testWorld = this.w.start(200,400)
                   && this.w.bigBang(0.3);
```

To erase earlier blobs, we would also draw some background for the world.

Time-Driven World

There is only a small addition needed to allow the program to perform specific tasks at each tick of the clock. The clock speed is determined when the program starts, and then at each tick, the method designed to respond to the timer events is invoked. We decide that at each timer tick, the blob should move in a random distance, no more than 5 pixels in any direction, unless its center is outside of the canvas bounds, at which point the game stops.

We first add the needed methods to the `class Blob`. We need a method to produce a `Blob` moved a random distance, and a method that determines whether this `Blob` is within the given bounds. Here are the purpose statements and contracts for these methods:

```
// produce a new blob moved by a random distance < n pixels
Blob randomMove(int n){ ... }

// is the blob outside the bounds given by the width and height
boolean outsideBounds(int width, int height){ ... }
```

Exercise: Design these two methods. You should use the following helper method to produce the next random number:

```
// to generate a random number in the range -n to n
int randomInt(int n){
  return -n + (new Random().nextInt()) % (2 * n);
}
```

We now need to add the method `onTick()` to the class `TimerWorld`. This method is invoked on every tick, in a manner similar to the way the `onKeyEvent` method is invoked when a key is pressed.

```
// what happens when the clock ticks
World onTick() {
  // if the blob is outside the canvas, stop
  if (this.blob.outsideBounds(this.width, this.height))
    return this.endOfWorld();

  // else move the blob randomly at most 5 pixels in any direction
  else
    return new TimerWorld(this.blob.randomMove(5));
}
```

Games

You can now design your own computer game that responds to the key events and/or timer clicks. Here are some possibilities:

- **Worm Game:** A worm (consisting of a head and a list of segments) moves in 'its' direction on each tick, unless the user selects s different direction through the key stroke. A food morsel appears at random in the play area. If the worm eats the food, it grows by a new segment. The game ends when a worm either runs into the wall, or it 'eats itself', i.e., the head attempts to move is such way that it would eat a part of itself.
- **UFOs:** An UFO is falling from the sky - moving slightly sideways as the wind blows. The user can move a gun platform left or right, and shoot a shot with the keystroke of letter 'x. Keep shooting, till you either hit the UFO, or the UFO lands on the earth. Add more shots, more UFOs, etc.
- **Ant Game:** An ant travels through the play area controlled by the arrow keys. As it moves, is looses weight from hunger. When it finds food (a number of food morsels appear at random in the play area), it grows bigger. The game ends when the ant is too small to live, or get too big to move. (You choose what is too small or too big). It can also end when the ant hits the wall.
- **Star Thalers:** Star Money, Star Thalers by the Grimm Brothers

  There was once upon a time a little girl whose father and mother were dead, and she was so poor that she no longer had a room to live in, or bed to sleep in, and at last she had nothing else but the clothes she was wearing and a little bit of bread in her hand which some charitable soul had given her. She was good and pious, however. And as she was thus forsaken by all the world, she went forth into the open country, trusting in the good God.

  Then a poor man met her, who said, "Ah, give me something to eat, I am so hungry."

  She handed him the whole of her piece of bread, and said, "May God bless you," and went onwards.

  Then came a child who moaned and said, "My head is so cold, give me something to cover it with."

  So she took off her hood and gave it to him. And when she had walked a little farther, she met another child who had no jacket and was frozen with cold. Then she gave it her own, and a little farther on one begged for a frock, and she gave away that also.

  At length she got into a forest and it had already become dark, and there came yet another child, and asked for a shirt, and the good little girl thought to herself, "It is a dark night and no one sees you, you can very well give your shirt away," and took it off, and gave away that also.

And as she so stood, and had not one single thing left, suddenly some stars from heaven fell down, and they were nothing else but hard smooth pieces of money, and although she had just given her shirt away, she had a new one which was of the very finest linen. Then she put the money into it, and was rich all the days of her life.

Exercise: stolen fair and square from TS!2 workshop
Develop a game program based on the story of "Star Money, Star Thalers."

The program should consume a natural number and drop that many thalers (from the top of the world) on the girl (at the bottom of the world), one at a time. The thaler should move randomly to the left or right and downwards, but should always stay within the boundaries of the world (canvas). The girl should react to 'left and 'right keystrokes, moving a moderate number of pixels in reaction but always staying completely within the boundaries of the world.