miami university
jacob tonksi
tonskij@muohio.edu

Code the Complexity

Here's an example designed to introduce the way in which complexity can be quickly achieved through code.

STEP 1: GET SET UP.

Download and install Processing from www.processing.org.

Download and unzip the MovingType example. You'll have a folder named MovingType, and inside, a file named MovingType.pde. Open this pde file in Processing.

If it's your first time looking at code, you'll see a lot of odd words and symbols. Don't worry. The goal of this example is to demonstrate similarities between code and more familiar language and symbols, so we'll simply ignore sections of the code and pay attention to a few specific places.

STEP 2: WHAT DOES THIS DO?

Press the play button, top left corner of the window. Type a few keys and watch what happens. Not too exciting yet, but let's see what we can do if we make a few changes...

STEP 3: A QUICK TOUR OF SOME CODE HIGHLIGHTS.

Let's read bits some of the code...

Throughout the code, you'll see pairs of forward slashes: //

The pair of slashes tell Processing to ignore all remaining text on the line. This is called a comment, and is useful for two things: making notes about what's going on in the code, or "disabling" a command, easily re-enabled later by simply removing the //.

(Note that the line number of your cursor is displayed in the very bottom left corner)

Look for the following text on line 13:

void setup() {

This is the "setup" section of our sketch. Everything between the left curly brace at the end of the line and the right curly brace all alone on line 29 happens just once at the very beginning of the program running. This allows us to load fonts, images, or calculate some values that we may need while the program runs.

Note that line 15 says:

size(800, 600);

This is a command that sets the size of our canvas. All commands are followed by a matching pair of parentheses followed by a semicolon, which functions just like a period at the end of a sentence when writing. Sometimes the parentheses contain a list of details about how to perform the command, separated by commas. Here, the two details specify the width and height of the canvas in pixels.

Now look on line 32 for:

void draw() {

The draw section ends at the } on line 60. This section of the code runs over and over to generate each frame of the animation, about 30 times a second.

On line 35 you'll see:

background(0,0,0);

This command fills the entire screen with one color. The three details, or parameters, for the command here are the red, green and blue values that comprise the desired color. These values each range from 0 - 255, 0 being dark and 255 being the brightest value in each channel. background(255, 255, 255) fills the screen with white.

On line 47, you'll see a more complicated command that begins with the word "for". Here again you'll see a left curly brace - this one corresponds to a right curly brace on line 59. All of the commands between those two braces will occur repeatedly - once for each letter we've typed. Don't worry about why or how - just notice that we can type a command once and have it apply multiple times by simply placing it inside of what's called a loop. Notice the command:

letters[i].drawLetter();

Here, letters is the name of the list of letters we've typed and [i] is a name which refers incrementally to each number as the command is repeated. drawLetter() tells that letter to draw itself on the screen.

Note the commented commands which look somewhat similar on lines 52-55.

STEP 4: MAKING SOME CHANGES...

Each time you make a change to the code, press run again to see the changes take effect.

4.1 Try removing some of the comment //'s on lines 52-55 and re-running the sketch. Start by enabling the fallDown() command, then spinAround().

4.2 Try removing the // before either of line 36 or lines 38-39, and place a new // before the background command on line 34 (the latter step here is important). Rerun the sketch.

The fill command on line 41 lists 4 numbers. In order, they refer to the red, green, blue and opacity properties of the current fill color, which is used to fill the rectangle drawn on line 42 (which is drawn from the top left corner, 0, 0, across the width and height of the screen). This acts like a semi-transparent background command. Imagine adding a piece of colored acetate over the canvas each frame, tinting everything below it.

Now let's look at how each letter's initial position is established.

Line 63:

void keyPressed() {

This section of the code is executed by Processing each time we press a key.

Look at line 84:

curX = curX + letterStep;

This adds a certain number of pixels to the value stored under the name curX (short for current X). That amount depends on the value stored under the name letterStep. This value is set to 20 on line 8. Try changing this.

Next, let's have each new letter we type show up at the current mouse position. On lines 93-94, you'll see:

```
//curX = mouseX;
//curY = mouseY;
```

Uncomment those lines (remove the //'s). Now, no matter what value curX has, it will get replaced here with mouseX (same for Y), which always stores the current mouse x position. When we create a new LetterForm based on the key that was pressed (lines 97-102), curX will store the current mouse X. Run this, and move your mouse around as you type.

STEP 5: DIGGING A LITTLE DEEPER.

At the top of the window, you'll see a tab called LetterForm. Let's look at the code on that tab by clicking on it.

Scroll down and near the bottom of the page you'll see on line 66:

```
void fallDown() {
```

Line 68 says:

```
yPos = yPos + yMovementSpeed;
```

Here, each time we tell a letter form to fall down, its y position has a value added to it - a value stored under the name yMovementSpeed. Up near the top of the page on line 16 you'll see:

```
float yMovementSpeed = 2;
```

Try changing this value and running the sketch anew.

Back down at line 71 we see that not only is yPos changed each time, but so is yMovementSpeed. This is what creates the acceleration of the letter as it falls. Each LetterForm stores its own yMovementSpeed, so

each letter accelerates independently.

The following lines of code on lines 74-77 say that if the letter's y position is greater than the height of the screen, it's y position should be set to 0, the top of the screen, and its speed set to 1.

Try playing with some of the number values you see in the code here. Note that most of them imply motion in units of pixels per frame, not inches or seconds.

Try placing a comment // in front of the line that creates acceleration. This will change the motion of the sketch noticeably if you are also telling the letters to fallDown back in the draw() section of the MovingType tab.

Try combining various behaviors of falling, spinning and orbiting with various background fill methods.