

JAN/FEB **2024**  
ISSUE **22.1**

XDEV.MAG.COM

**xdev**<sup>TM</sup>

THE MAGAZINE FOR XOJO<sup>TM</sup> USERS

# SANDSTORM

SIMULATING FALLING SAND WITH XOJO





# READY TO DEPLOY YOUR XOJO WEB APP?

*Developing, testing and deploying your  
app has never been so easy.*

**TRY XOJO CLOUD TODAY!**

[XOJO.COM/CLOUD](https://xojo.com/cloud)



JAN/FEB 2024

ISSUE 22.1

XDEV.MAG.COM

# xdev

THE MAGAZINE FOR XOJO™ USERS

## FEATURES



### 12 Faster, Xojo, Faster!

BY MARC ZEEDAR

We got a new faster Xojo for Christmas, but just how much improved improvement will we see? Marc runs some tests.



### 20 Design Patterns Part 2

BY MARC ZEEDAR

Our design pattern series continues with a look at the Decorator (a.k.a. Wrapper) pattern. It allows you to dynamically change the behavior of objects at run-time by wrapping one object with another.



### 36 Raspberry Pi Electronic Fun Part 5

BY EUGENE DAKIN

Getting information from your Raspberry Pi is important for debugging. Today we're going to demonstrate how to retrieve GPIO info, such as the state of various pins.



### 44 Sandstorm

BY MARC ZEEDAR

Marc got a "falling sand" desktop toy and used it as inspiration to simulate it in Xojo. Just how hard is it creating digital falling sand? Not hard at all, it turns out.

*xDev Magazine* (ISSN 1540-3122) is the ultimate source for tutorials and advanced techniques for programming with the Xojo development environment. *xDev Magazine* is published bi-monthly (six times per year) by DesignWrite, 14888 NW Willis Road, McMinnville, OR 97128.

**Subscription Rate:** Annual subscriptions are \$49.99 (PDF) and start at \$99.99 for printed issues (<http://www.xdevmag.com/subscribe/>).

**Customer Service:** <http://www.xdevmag.com/support.shtml> (Address changes: <http://www.xdevmag.com/addresschange.shtml>). **Back Issues:** <http://www.xdevmag.com/orders.shtml>

JAN/FEB 2024

ISSUE 22.1

XDEV.MAG.COM

# xdev™

THE MAGAZINE FOR XOJO™ USERS  
COLUMNS

- 5 Source Code:** A word from the Publisher MARC ZEEDAR  
*A new year.*
- 65 Beginner's Corner:** For those getting started MARC ZEEDAR  
*Nonobvious "obvious" solutions.*
- 73 Xojo Talk:** Thoughts on Technology PAUL LEFEBVRE  
*2023r4, handling exceptions.*
- 77 MBS Spotlight:** Learning the MBS Plugins STEFANIE JUCHMES  
*Parsing emails.*
- 82 Best of the Web:** Programming articles you may have missed MARC ZEEDAR  
*P versus NP, learning about learning, running a generative AI locally, and the ad-free internet.*

## NEWS, REVIEWS, ETC.

Xojo News . . . . .	8
Xojo Calendar . . . . .	11

## ADVERTISER INDEX

Xojo, Inc. xojo.com . . . . .	2
MBS Plugins monkeybreadsoftware.de . . . . .	25, 81
xDev Archive Book 21 Preorder xdevmag.com . . . . .	35
App Wrapper 4 ohanaware.com . . . . .	48, 75
xDev Archive Books xdevmag.com . . . . .	55
xDev Magazine xdevmag.com . . . . .	84

# How to Download Source Code

**Article Resources:** <http://www.xdevmag.com/browse/>

The “browse” website gives every *xDev* article a *permanent web page* where you can find links to downloads, updates and corrections, and more. Note that downloads include a modification date. Every article in *xDev Magazine* includes an “XD number” at the beginning presented like this: **XD#10234** To retrieve an article’s resources, follow these steps:

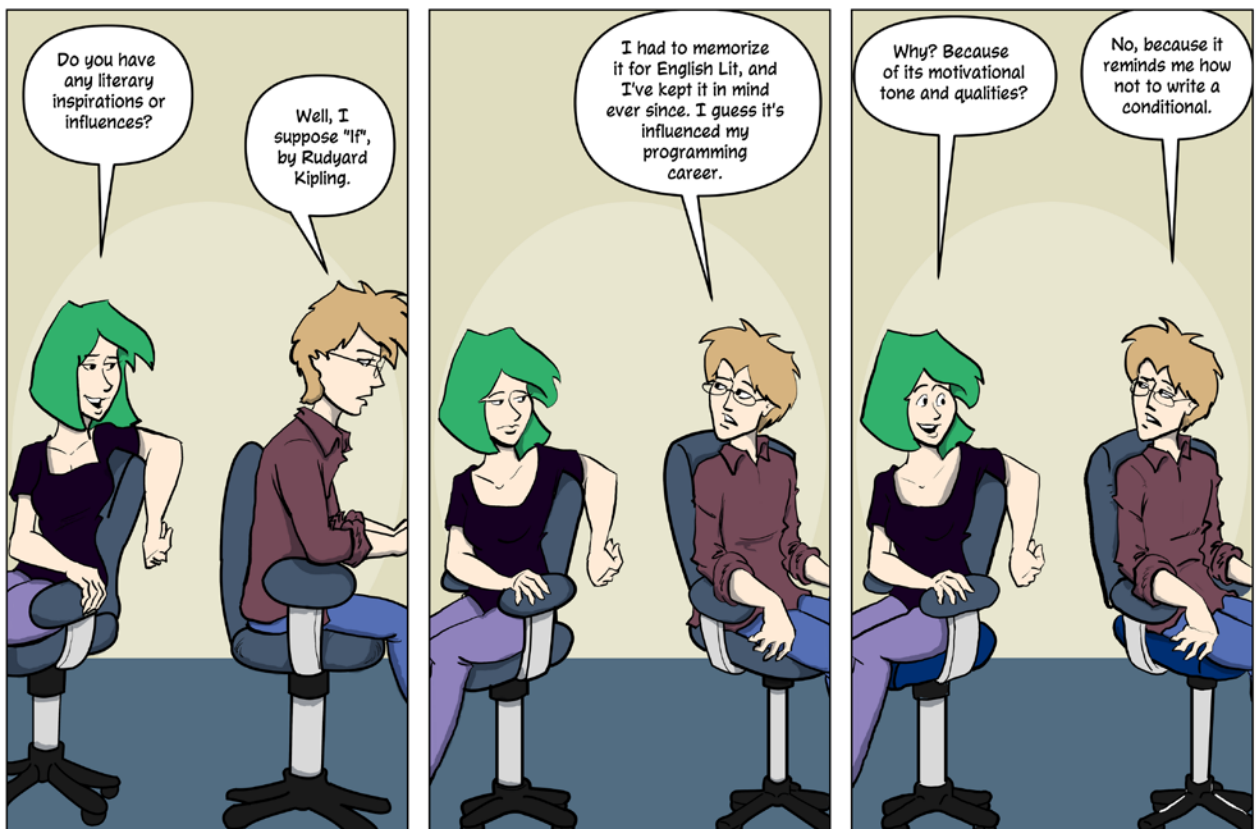
**Step 1:** Go to the *xDev* website (<http://www.xdevmag.com/browse/search.shtml>).

**Step 2 (find article by number):** In the **Get Article** search field type in the *XD number* of the article you need.

**Alternate Step 2a (find article by issue):** Find the issue you’d like to Browse and click on the link for that issue’s Table of Contents.

**Alternate Step 2b (download all resources):** On an issue’s Table of Contents page, there’s a link to a file which includes *all the downloads* for a particular issue in one large archive.

## HACKER by Dan Wilson <hacker@xdevmag.com>



by Marc Zeedar

# Sandstorm

## Simulating Falling Sand

---

**R**ECENTLY I purchased a little “falling sand” desk toy (see Figure 1). I’m a sucker for gizmos like this. I can watch lava swirling or sand falling forever and always find it fascinating.

As I was playing with my new toy, I started wondering if it would be possible to simulate falling sand on my computer. Well, I knew it would be possible—computers can simulate anything with enough data and programming—but I wondered if *I* could do it. You see, I’m the opposite of a math wizard and usually simulating physics requires a lot of calculations.

### Creating Sandstorm

I dove in anyway, launching Xojo and tinkering. Right at the start I decided to do my simulation pixel-for-pixel—in other words, each pixel in a canvas would be a grain of sand. After all, grains of sand are tiny, so pixels made sense.

#### AT A GLANCE

**XD#22105***Target Reader:***Beginner***Source Code:***Yes***About the Author:*

**Marc taught himself programming in high school when he bought his first computer but had no money for software. He’s had fun learning ever since.**



**Figure 1:** *Marc's new falling sand toy.*



**Figure 2:** *Swirling sand would be much more complicated to simulate.*

I did a *little* internet research for algorithms, and as usual much of what I found was overly complicated, designed for video games, and was in other languages that would require understanding them in order to translate the code to Xojo. Though nothing was Xojo-ready, I did get a glimmer of a direction or two and started playing.

Already I knew I wasn't going to do anything too complicated. For instance, I would just have regular vertical (down) gravity, not gravity at various angles, and though my desktop toy includes water and air bubbles in the frame that makes the sand swirl and dance in unusual ways (see Figure 2), I wasn't about to try and simulate that!

So I began with just a simple "sand falls straight down" simulation. There are no currents or wind to change the direction of the sand. It turned out that this is quite easy to do and the results weren't half bad (see Figure 3).

**Side Note:** I did run into some weirdness with a bug that was causing my sand picture to not draw properly—weird extra lines at the edge amongst other oddities. That turned out to be a mistake in my code.

Though I'm well-aware that arrays start with a 0 index, I forgot that `rgbSurface.pixel` works the same way. I had specified that my picture be 360 pixels wide, but I was accessing pixels that didn't exist with code such as





**Figure 3:** *An early, primitive version of Sandstorm demo.*

`sandPic.rgbSurface.pixel(360, 360)`, causing strange phenomena (though no error messages).

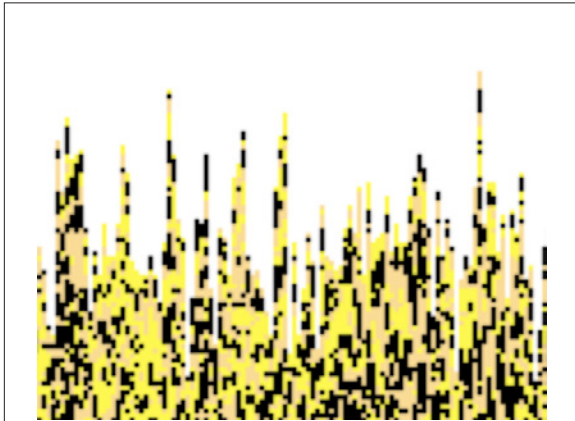
Once I constrained my pixel processing to the 0–359 range of my picture, my “sand” started behaving properly.

### *The Basic Premise*

I’ll get to some actual code in a minute, but first a quick overview of what I did. There are several key elements:

- A custom canvas class called `sandCanvas` is used to display the falling sand.
- Animation of the falling sand is triggered via a timer which calls `sandCanvas.fall` for each frame.



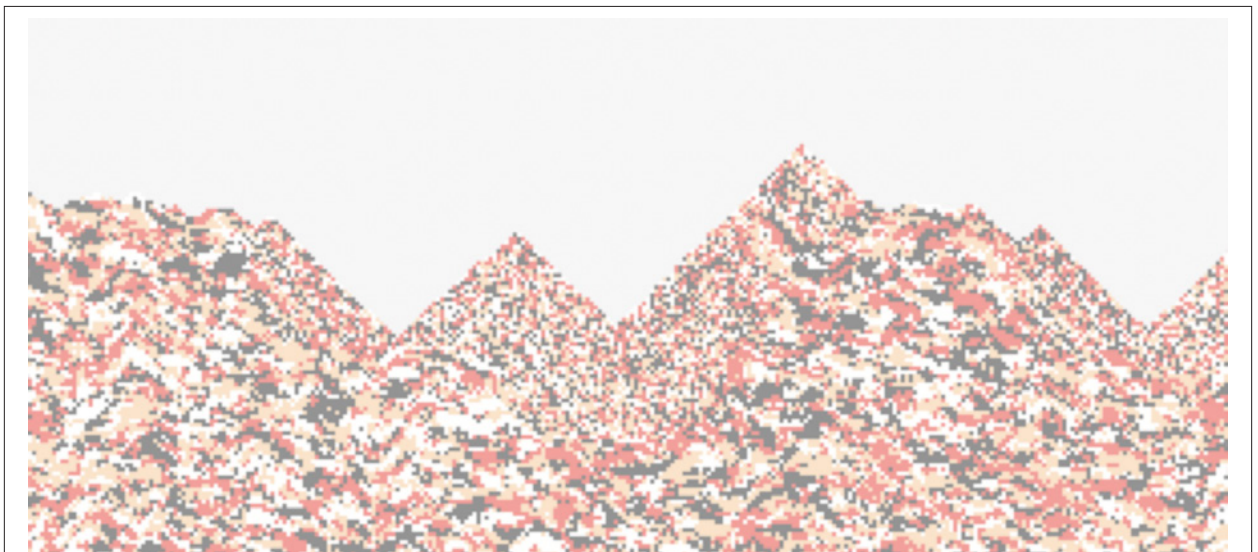


**Figure 4:** *With no horizontal movement, sand pixels just pile up in thin vertical lines, which isn't the way real sand behaves.*

- Drawing is done on a picture object, `sandPic`, which is displayed via the `sandCanvas.paint` event.
- Several colors of sand are stored in a `sand-ColorList` array and a certain amount of “sand” is randomly scattered across the entire canvas during initiation. Once the “Start Falling” button is clicked, animation begins and the grains of sand fall straight down.
- We'll add the ability to add some random barriers onto the display to create some obstacles for the sand to encounter on its way down, for a more interesting visual effect.

That's pretty much the entire demo, though there are some subtleties and some additional features I'll explain later.

Oh, and one more thing that's significant. In the real world, my sand toy is circular and can be rotated to cause the fallen sand to end up on the side or top and fall again. For Sandstorm, I added a “flip” button that turns the canvas 180 degrees and the sand then falls from wherever it landed the previous time.



**Figure 5:** *With random horizontal movement added, the simulated fallen sand creates realistic dunes and mountain peaks.*

### *The Problem with Down Gravity*

Making the sand fall straight down is simple. The algorithm is pretty much: “Is the pixel under me empty? If so, move me there.”

That’s it. Very easy!

The problem is that’s not very realistic. The sand just piles up in a single pixel wide vertical line (see Figure 4). In real life, a few grains of sand might stack like that, but eventually they’d start falling off to the left and right.

The solution is to add some variation to the falling. The algorithm I came up with is so simple, but astonishingly effective. All I do is wait until there are three grains of sand in a stack, and then I randomly tumble the next falling grain either left or right. That’s it!

The effect is so cool: sand tumbles down and forms triangular peaks, gradually pours off the sides, and stacks in realistic hills and valleys. Eventually it reaches an equilibrium and the sand won’t move any more, exactly like real sand! (See Figure 5.)

So how do we code all this? Let’s begin!

### *Initializing Sandstorm*

As mentioned, we’ll begin by creating a subclass, `sandCanvas`. Create a new class (use the Insert menu and choose Class) and set its super to `desktopCanvas`.

Make your life easier with the  
Mac App Store & Gate Keeper.



*Puttin' apps in  
the App Store  
since 2011*



Download a FREE trial today  
[ohanaware.com/appwrapper](http://ohanaware.com/appwrapper)

Add the following properties:

```
Public Property barrierColor As color = &c000000
Public Property falling As boolean
Public Property r As random
Public Property sandColorList() As color = &c000000
Public Property sandPic As picture
```

I'll explain how these are used as we encounter them in our code. For now, let's begin with our initiation routine, which is about 80% of this app. First, though, add an Opening event to the class and call init in its code:

```
Sub Opening() Handles Opening
    init
End Sub
```

This ensures that init is called when the app runs and the canvas is first opened.

Now create a new method in sandCanvas called init and put in the following code:

```
Public Sub init()
    sandColorList = array(&c82828200, &cFF868600, &cFFDDBC00, &FFFFFFE00)
    sandPic = new picture(371, 371)
    var d as new date
    r = new random
    r.seed = d.totalSeconds
    var c as color
    for x as integer = 0 to 370
        for y as integer = 0 to 370
            sandPic.rgbSurface.pixel(x, y) = kWhite
            if previous(x, y, c) or above(x, y, c) then
                // Increases odds it's the same color.
                if rnd > .60 then sandPic.rgbSurface.pixel(x, y) = c
            else
                if rnd > .8 then sandPic.rgbSurface.pixel(x, y) = randomColor
            end if
        next y
    next x

    // Put some obstacles on the canvas.
    if keyboard.optionKey then
        barrierColor = xoyoLogo.rgbSurface.pixel(100, 50)
        sandPic.graphics.drawPicture(xoyoLogo, 50, 50)
```

```

else
    var x as integer = r.inRange(50, 300)
    var y as integer = r.inRange(50, 200)
    barrierColor = color.brown
    sandPic.graphics.drawingColor = barrierColor
    sandPic.graphics.fillRect(x, y, 50, 20)

    x = r.inRange(50, 300)
    y = r.inRange(50, 200)
    sandPic.graphics.fillRect(x, y, 50, 20)

    x = r.inRange(50, 300)
    y = r.inRange(50, 200)
    sandPic.graphics.fillRect(x, y, 50, 20)
end if
me.refresh
End Sub

```

The first thing this does is set up the colors of the sand we are going to use (these are a southwestern set, of reddish-peaches, gray, and tan) and store them in the `sandColorList` array. If you want to use different colors, this is where you'd set those.

We then set up our `sandPic` picture property. Note that I've hard-coded the picture size in this app. That is probably not the best way to do this, as it means the canvas can't be resized and still work properly, but it's fine for a demo. If you wanted, you could replace these size values throughout the app with a constant or a calculated variable that adjusts to the actual size of the canvas.

Next, we set up our random property and seed it.

Then we get to the good stuff: creating sand. We set up two loops for the `x` and `y` dimensions of `sandPic` and step through every pixel of the image. You'll note that we first establish a default pixel color of `kWhite`. That's a constant we need to add to the class, so let's do that now:

```
Public Const kWhite as Color = &cFFFFFFee
```

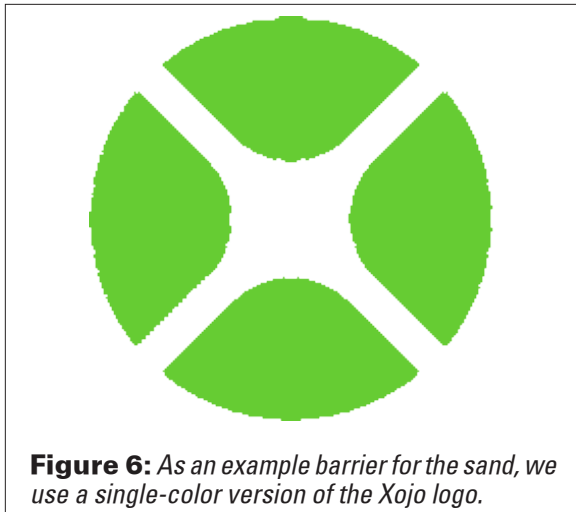
Notice the color is set to white, but I've added an `EE` alpha channel at the end, which sets it to transparent. That gives us a picture we can see through and lets the background behind our sand show through. More on that later.

Going back to the init routine, we call two functions on each pixel, `previous` and `above`. They are very similar routines and look like this:

```
Public Function above(x as integer, y as integer, byref c as color) As boolean
    if y > 0 then

```





```

    c = sandPic.rgbSurface.pixel(x, y - 1)
    return c <> kWhite
end if
End Function

```

```

Public Function previous(x as integer, y as integer, byref
c as color) As boolean
    if x > 0 then
        c = sandPic.rgbSurface.pixel(x - 1, y)
        return c <> kWhite
    end if
End Function

```

What these do is return true if the pixel before (one to the left) or just above are not white (kWhite). The actual color found is returned in the c property, which is a byref parameter.

The reason I did this is I wanted the sand to somewhat clump together common colors. This code increases the odds of the same sand color being picked instead of being totally random. If I didn't do this (and if the random routine was truly random), the distribution of colors would be fairly even throughout the sand display. This helps make the sand be distributed more unevenly. Right now I have this line of code:

```
if rnd > .60 then sandPic.rgbSurface.pixel(x, y) = c
```

This gives us a 50/50 chance that the color we pick will be the same color as the grain of sand to the left or above the new pixel. If you want to experiment, you can change the .50 to a lower or higher number and see how that changes the sand patterns produced.

If there are no sand pixels to the left or above, then the new pixel has a 20% chance of being any color via this code:

```
if rnd > .8 then sandPic.rgbSurface.pixel(x, y) = randomColor
```

The .8 in that line essentially controls how much sand is in the display. A smaller number would increase the sand—a larger number means less chance each new pixel would be sand (if it's not a sand pixel, it's just kWhite and is transparent).

The randomColor function just selects a color from sandColorList and returns it:

```

Public Function randomColor() As color
    var c as integer = r.inRange(0, sandColorList.lastIndex)
    return sandColorList(c)
End Function

```



**Figure 7:** A faded background of an apartment is used behind the sand to make our simulation look more like a real glass display toy.

At this point in the `init` routine, all our sand is colored and scattered on the display. The final part of this code sets up some obstacles for the sand to hit.

For this demo I just manually coded in some brown rectangles that are randomly placed on the `canvas`. I had an idea for a future improvement where the user can drag and move the blocks around, and/or draw their own, but I didn't implement that.

I also wanted to demonstrate how to bring in external graphics to use as a barrier. For that example, I use a single-color version of the Xojo logo (see Figure 6). It's vital the logo only contain one color, because we sample that color with this line of code:

```
barrierColor = xojoLogo.rgbSurface.pixel(100, 50)
```

This sets our `barrierColor` and is used to tell if the falling sand hits an obstacle. If there are shades of green in the Xojo logo, for instance, those won't *exactly* match the `barrierColor` and will be treated as grains of green sand and will drop with all the other sand!

(Yes, I found this out the hard way when the logo obstacle "melted" in my trials.)

To add the barrier graphic, we just draw it onto `sandPic` with the `drawPicture` command.

Note that for the purpose of the demo, I have made the rectangular blocks the default option. If you want to test the Xojo logo barrier, you need to hold the Option (or Alt) key down during the `init` routine's execution.

The final part of `init` is to redraw the entire canvas (via `me.refresh`).

Speaking of drawing, let's add the `Paint` event where the sand is drawn:

```
Sub Paint(g As Graphics, areas() As Rect) Handles Paint
  if sandPic = nil then init
  if window1.cbBackground.value then g.drawPicture(apartment, 0, 0)
  g.drawPicture(sandPic, 0, 0)
End Sub
```

Wow, that's super-complicated, right? Really only the final line is needed to draw the sand. However, just for purity sake I do a `nil` check on `sandPic` in the first line.

The second line is a feature I added later in development to draw a background behind the sand, which just looks cool. I choose a public domain apartment picture which I blurred and faded so it looks like it's being seen at a distance through the glass of the sand toy (see Figure 7).

(Full disclosure: I added this after *xDev*'s fantastic artist, Jeff Quan, did it for this issue's cover art. He also created the wood frame I use in the app.)

I made this background optional via a checkbox control, just in case you prefer a plain white background (which does make the sand easier to see).

## Making Sand Fall

Now that we've got our sand ready, we need to make it fall. How do we do that?

There are two parts. First is the code that calculates the movement of the sand. The second part is triggering that via a `Timer`.

In `sandCanvas`, we add a method called `fall` which will be called for each frame of falling animation. This calculates one movement for each grain of sand in the image.

Note that I have not optimized this for speed. While my sand display is not very big (371 x 371 pixels), that's still over 137,000 pixels that have to be analyzed and adjusted for every frame (which for smooth animation should be 30 times a second).

On my fast M1 Max Apple Silicon machine the sand drops at a nice rate, but if you have a slower machine it may not be as smooth or as rapid. I'm not sure if that's a problem or not, but it's something to keep in mind. If needed, you could probably optimize this code to make it faster.

The main thing this code does is look at every pixel in `sandPic`, determine if it's a sand pixel or not, and if so, move it. To see if it is sand, we just check to make sure it isn't `kWhite` or `barrierColor`.

The movement algorithm is as I mentioned earlier: we check to see if the three pixels under this one are not `kWhite` (empty). If there's blank space underneath, we move the sand into that space. If it's blocked, we stack (leave it in place). If the stack is too tall (greater than three pixels), then we tumble it either left or right.

Once we have randomly selected a direction (left or right), we make sure that direction isn't at the edge of the frame (outside the picture), and we make sure that it's colored `kWhite` (empty). If everything's good, we set the color of the new pixel to the old pixel's value, and then set the old location to empty (`kWhite`).

One important thing about this process is that since we're erasing data (the old sand's color), we do our vertical loop through the picture in reverse (for `y` as integer = 370 downTo 0). In other words, the bottom sand falls first (which also makes sense in terms of real-world physics).

```
Public Sub fall()  
    // Now we animate the sand falling.  
    // We start at the bottom and drop sand if there's not a sand below it.  
    falling = true  
    var fallCount as integer  
  
    for y as integer = 370 downTo 0  
        for x as integer = 0 to 370  
            // Is there sand at current location?  
            if sandPic.rgbSurface.pixel(x, y) <> kWhite and sandPic.rgbSurface.pixel(x, y) <> barrierColor then  
  
                // Is there space below?  
                if sandPic.rgbSurface.pixel(x, y + 1) = kWhite then  
                    sandPic.rgbSurface.pixel(x, y + 1) = sandPic.rgbSurface.pixel(x, y)  
                    sandPic.rgbSurface.pixel(x, y) = kWhite  
                    fallCount = fallCount + 1  
  
                    // Is this too tall of a stack of sand?  
                    // If so, this piece should fall left or right.
```



```

elseif y < 368 and sandPic.rgbSurface.pixel(x, y + 1) <> kWhite and sandPic.rgbSurface.pixel(x, y + 2) <>
kWhite and sandPic.rgbSurface.pixel(x, y + 3) <> kWhite then
    var direction as integer = r.inRange(0, 1)
    if x = 0 then direction = 1 // Right.
    if x = 370 then direction = 0 // Left.
    if direction = 0 then direction = -1

    // Ensure there's space.
    if sandPic.rgbSurface.pixel(x + direction, y + 1) = kWhite then
        sandPic.rgbSurface.pixel(x + direction, y + 1) = sandPic.rgbSurface.pixel(x, y)
        sandPic.rgbSurface.pixel(x, y) = kWhite
        fallCount = fallCount + 1
    end if
end if
end if
next x
next y
me.refresh(true)

```

**Get xDev in book form!**



**All 21 Books Now Available!**

- Quality perfect-bound books
- 300-500+ pages each
- Color cover, black insides
- Complete issues of each year
- Discounts for current xDev subscribers

Order today at  
[http://www.xdevmag.com/orders\\_book.shtml](http://www.xdevmag.com/orders_book.shtml)

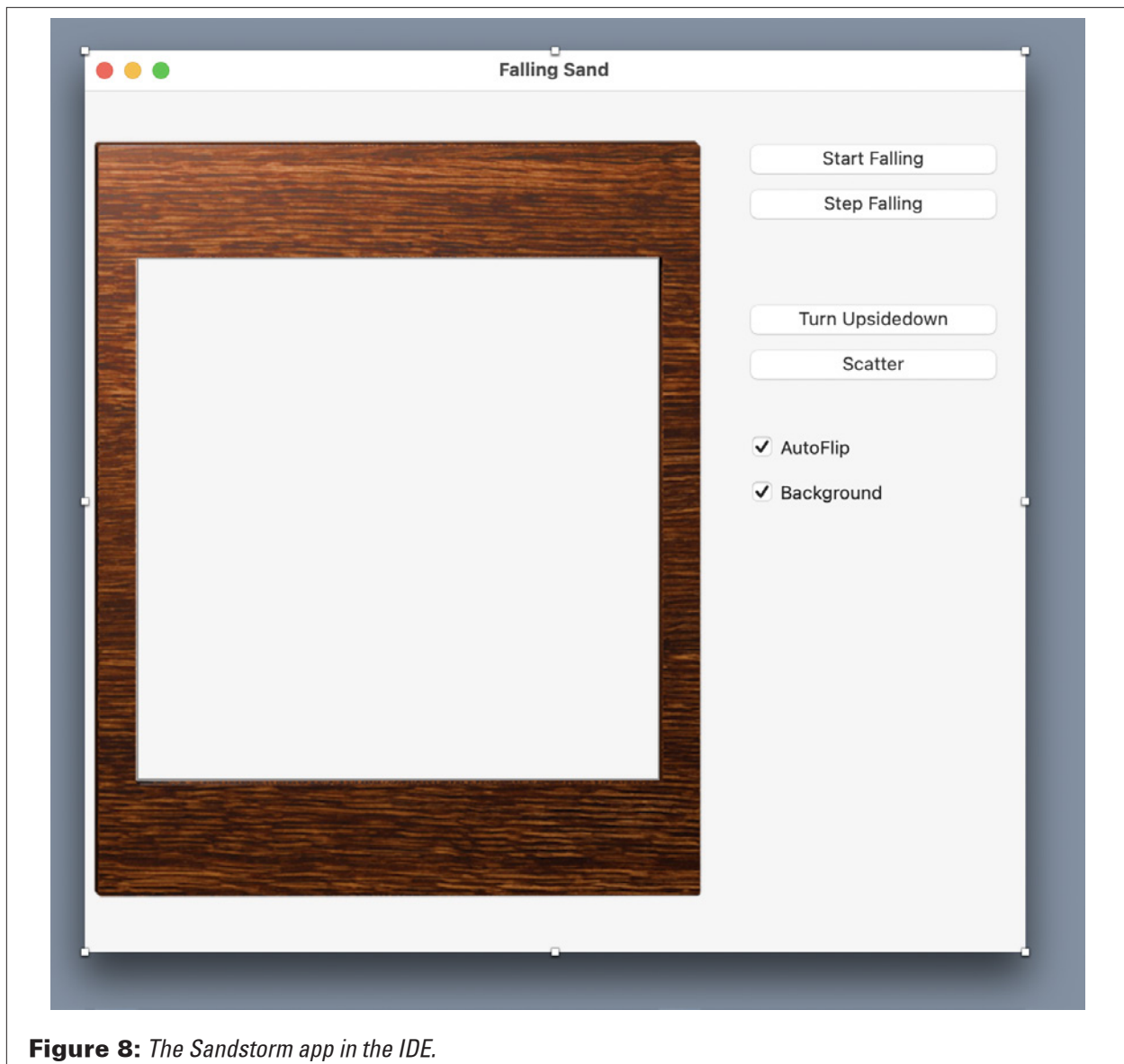
**twitter** 

**FOLLOW xDEV  
MAGAZINE ON  
TWITTER!**

Be notified of new issues, Xojo  
news and events, sales and  
special offers, and more!

**@xdevmag**

<http://twitter.com/xdevmag>



**Figure 8:** *The Sandstorm app in the IDE.*

```
if fallCount = 0 then falling = false
End Sub
```

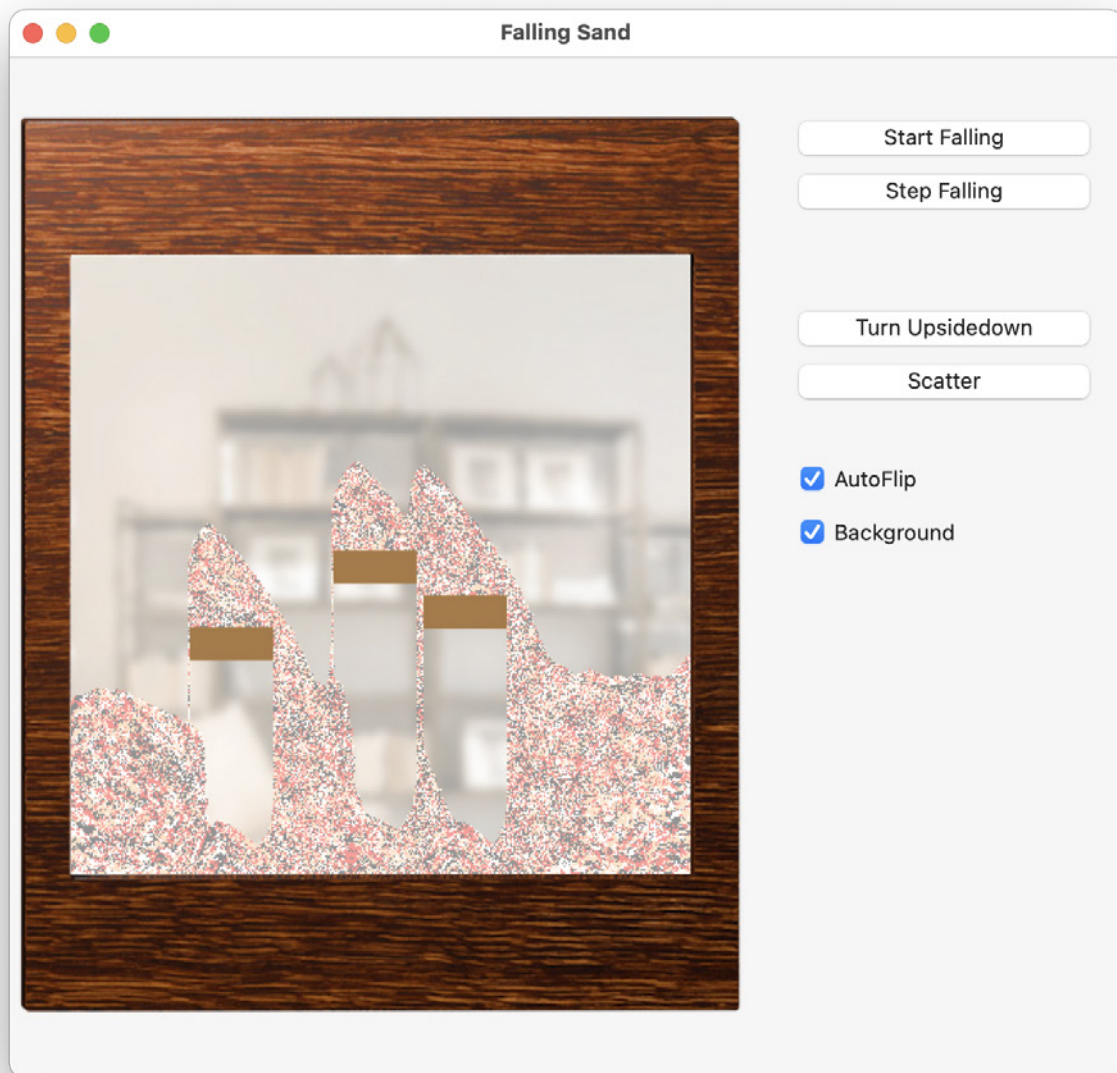
Two final bits about this code. You'll notice we have an integer property called `fallCount`—this counts the number of grains of sand that move during each animation pass. I do this so I can calculate when all the sand has fallen. Once no sand is moving after a call to `fall`, I know the sand is done. I can then set `falling`, a boolean property, to `false`. Other parts of my code then check that to determine if sand is still falling or not and act accordingly (by, for example, stop wasting clock cycles and calling `fall` when there's no sand to fall).

The `fall` routine handles the animation, but we still need to trigger it. To do that, we need to work on `Window1`, the main window of the app. I've set it up with several pushbuttons,

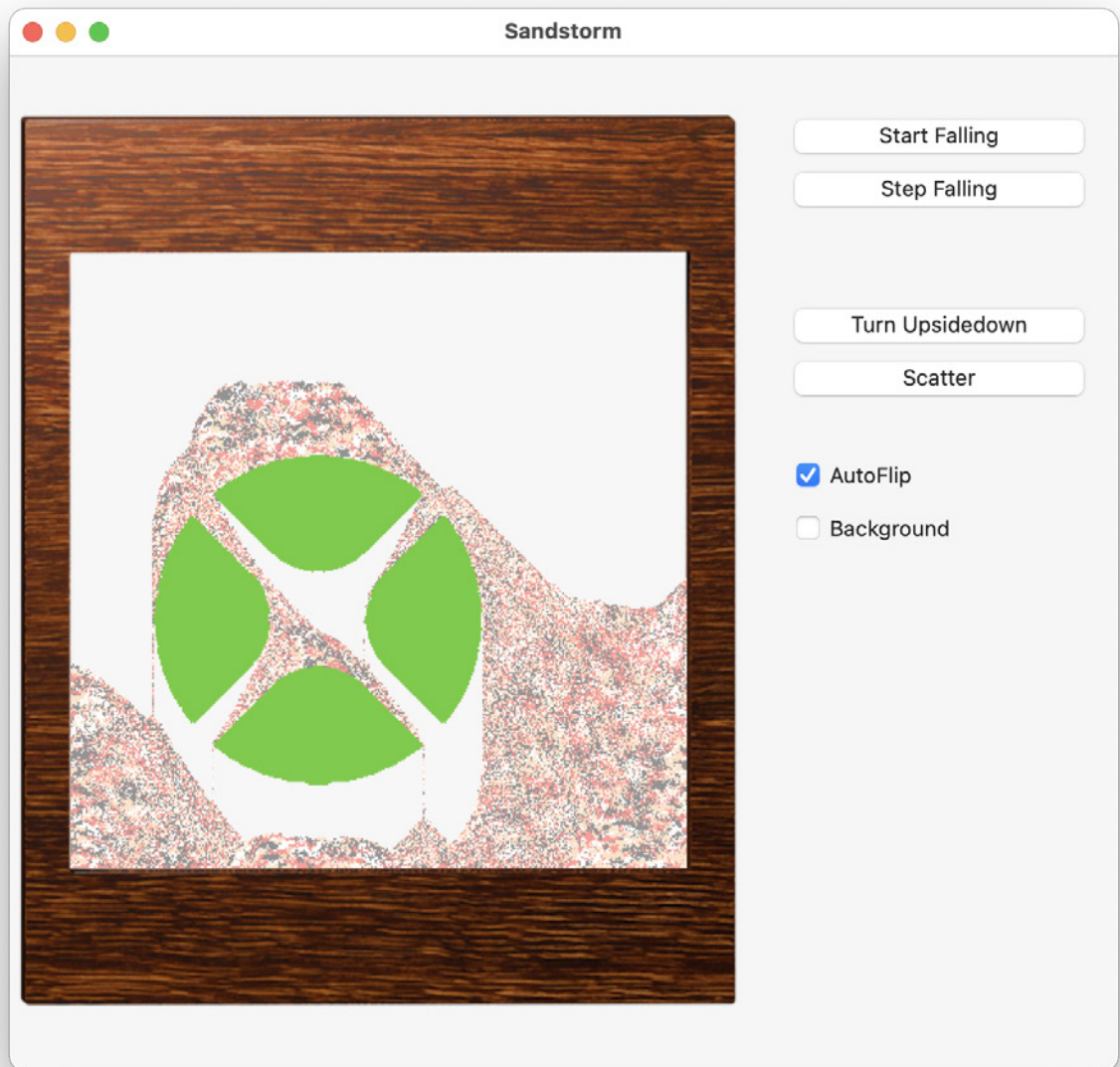
checkboxes, a wooden frame backdrop graphic, and an instance of `sandCanvas` which I've sized to 371 square and positioned into the opening of the wooden frame artwork (see Figure 8).

I've also added a timer called `fallTimer`, which is an invisible control but can be edited in the IDE. I set its period to 10 and it defaults to off. In its action event I add this code:

```
Sub Action() Handles Action
```



**Figure 9:** *Sand falling.*



**Figure 10:** *Sand falling around the Xojo logo.*

```
sandCanvas1.fall
if not sandCanvas1.falling then
  if cbAutoFlip.value then
    pbRotate.press
  else
    me.runMode = timer.runModes.off
  end if
```



```
end if
End Sub
```

You'll see the main thing this does is call `sandCanvas1.fall`, which triggers the sand animation. The rest of this has to do with options and configuration.

First, if the sand has stopped falling and the auto-flip checkbox is on, then we press the `pbRotate` button (which flips the sand upside down). If that option isn't set, we just turn off the timer (`me.runMode = timer.runModes.off`).

How does the sand flip work? `pbRotate` just calls `sandCanvas1.rotate180`:

```
Sub Pressed() Handles Pressed
    sandCanvas1.rotate180
End Sub
```

Of course, we haven't added `rotate180` to `sandCanvas` yet, so I guess we'd better do that!

It's actually pretty simple code. We first make a copy of the contents of `sandPic` into an array of `color`. Then we copy the backup back into `sandPic` in upside down order:

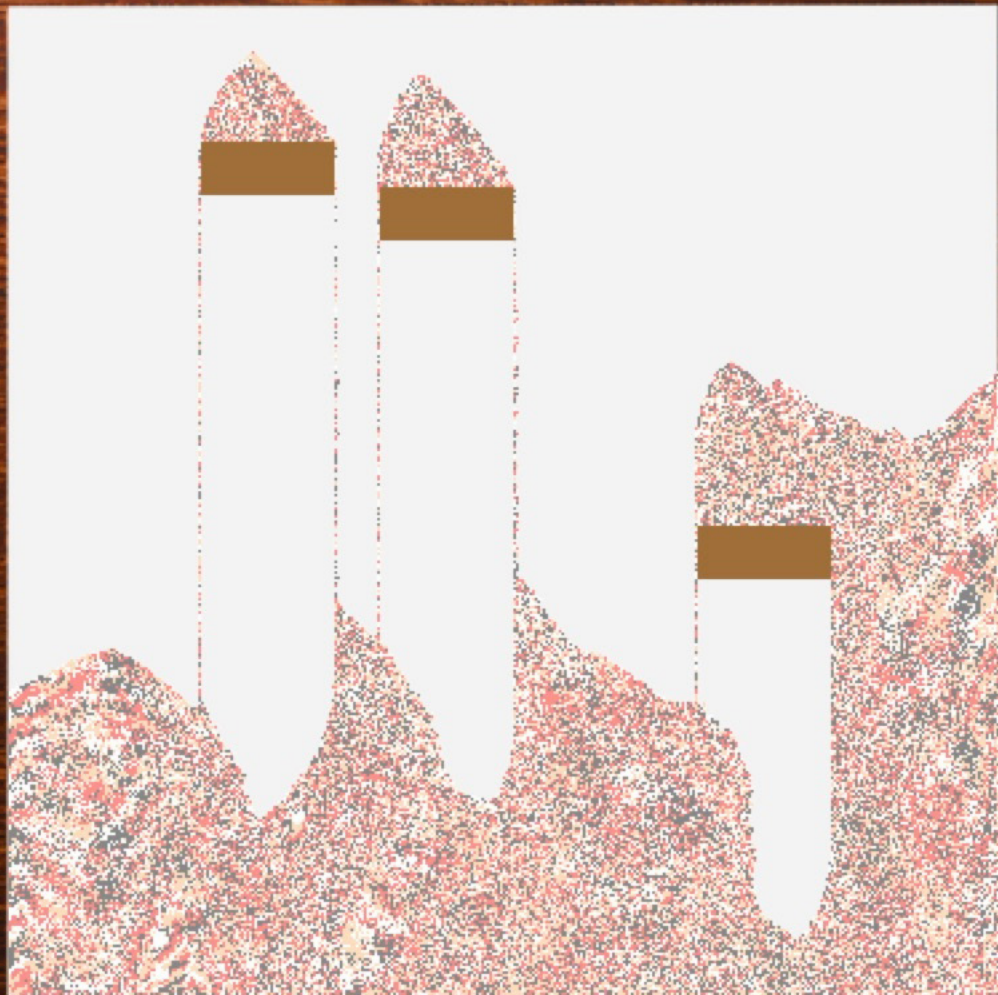
```
Public Sub rotate180()
    // First we make a copy of sandPic.
    var s(371, 371) as color
    for x as integer = 0 to 370
        for y as integer = 0 to 370
            s(x, y) = sandPic.rgbSurface.pixel(x, y)
        next y
    next x

    // Now we flip it upsidedown.
    for x as integer = 0 to 370
        for y as integer = 0 to 370
            sandPic.rgbSurface.pixel(x, 370 - y) = s(x, y)
        next y
    next x

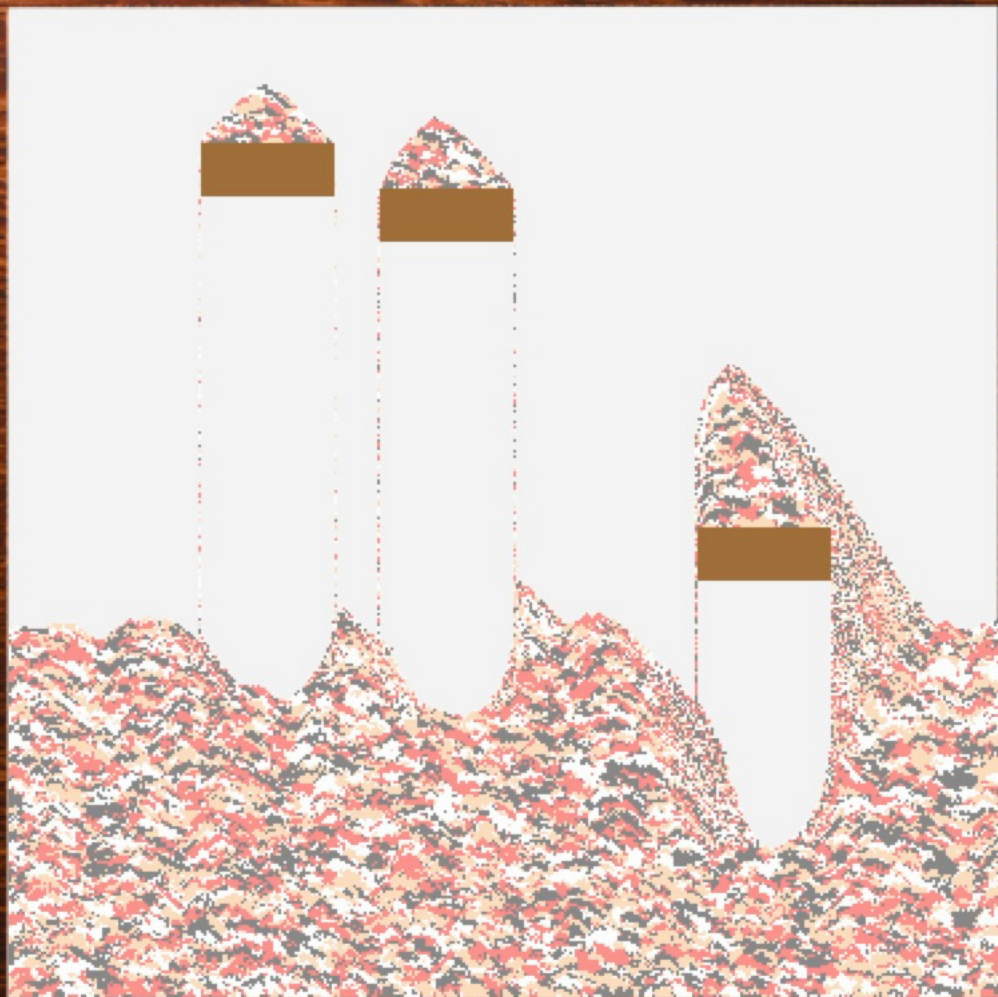
    me.refresh
End Sub
```

Bingo, now `sandPic` is upside down!

If you ran the program now you'd click the `pbStartFalling` button to start the animation and the falling sand might look similar to Figure 9.

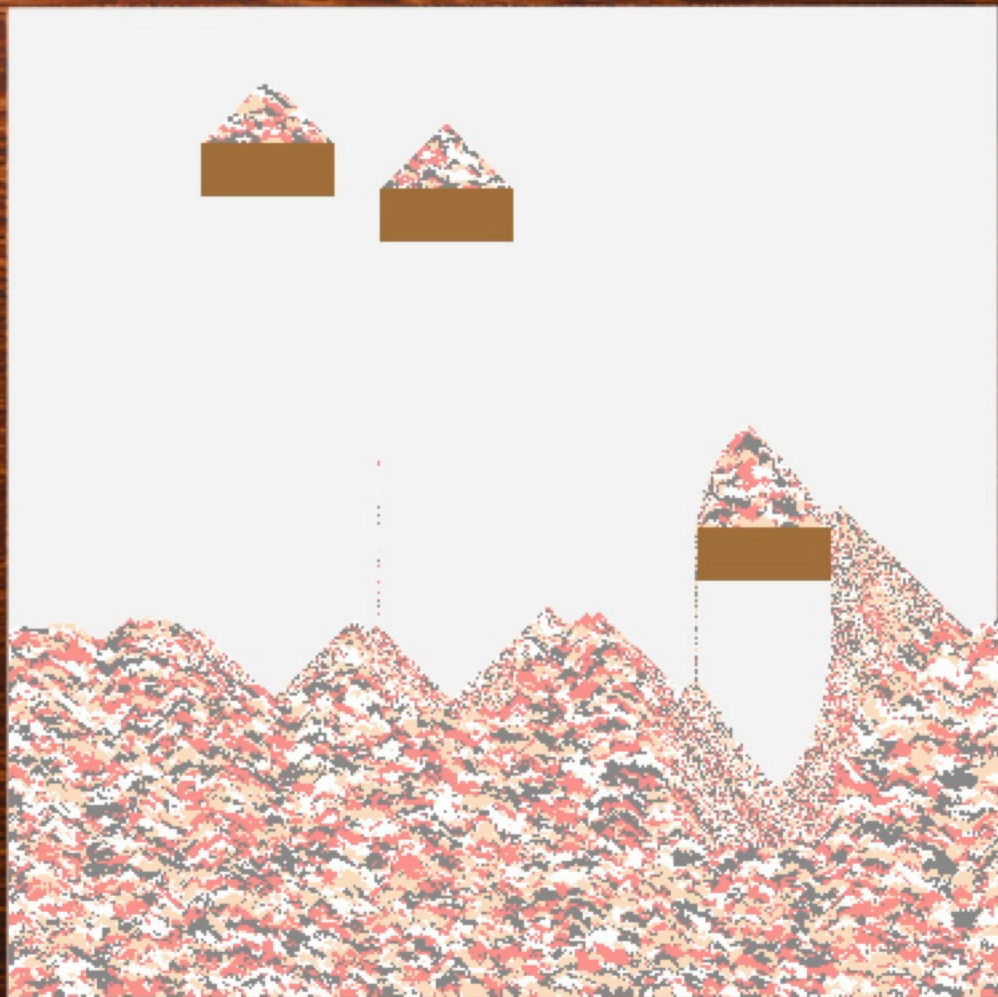


**Figure 11:** *Falling sand over time.*



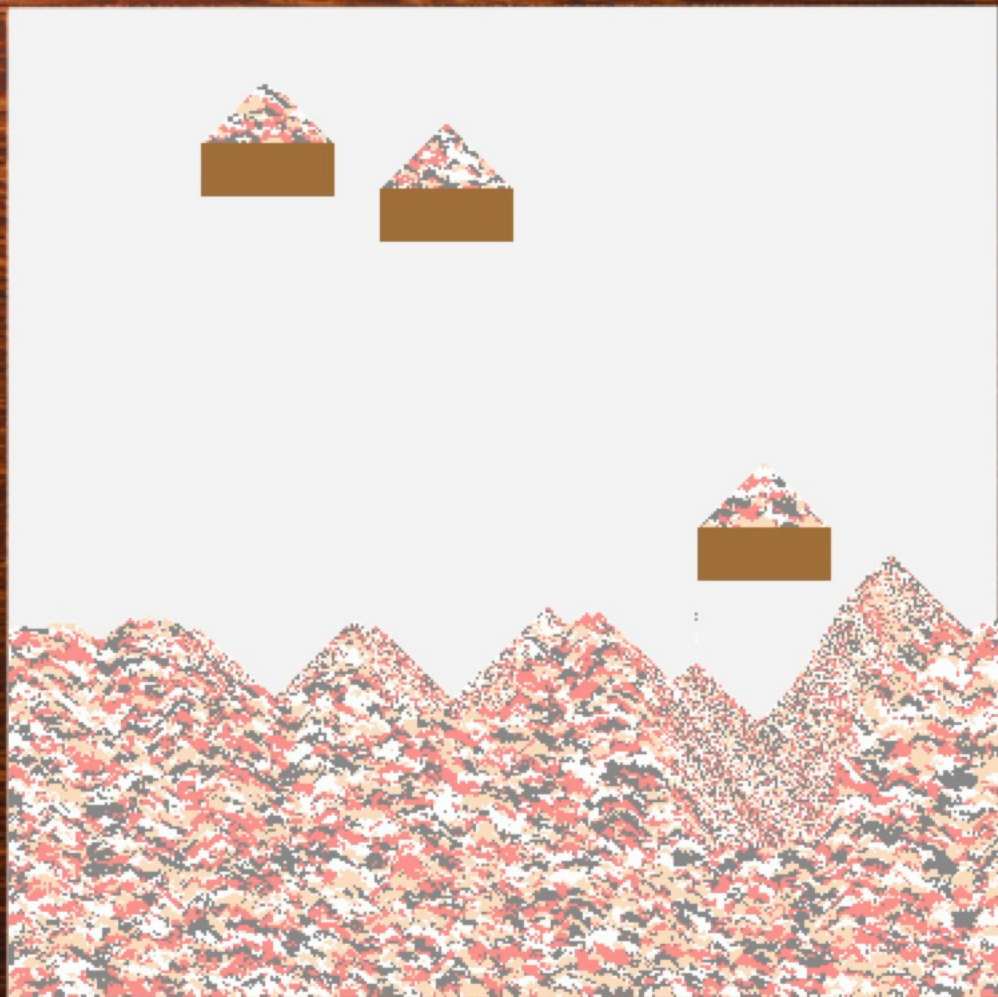
**Figure 12:** *Falling sand over time.*





**Figure 13:** *Falling sand over time.*





**Figure 14:** *Falling sand over time.*

## Wrapping Up

The rest of the controls on Window1 are pretty basic. pbScatter has a pressed event like this:

```
Sub Pressed() Handles Pressed
    sandCanvas1.init
End Sub
```

This just restarts the sand from scratch. Remember, if you click this button while holding down the Option (or Alt) key, it will scatter the sand with the Xojo logo as an obstacle (see Figure 10).

The pbStartFalling button:

```
Sub Pressed() Handles Pressed
    fallTimer.runMode = timer.runModes.multiple
End Sub
```

While it's not particularly useful during normal "play," for debugging purposes I added a "Step Falling" button (pbStartFalling). Instead of continuous falling sand, this lets you step through the falling so you can trigger the falling sand through each frame of the animation:

```
Sub Pressed() Handles Pressed
    fallTimer.runMode = timer.runModes.single
End Sub
```

That's pretty much it for Sandstorm. Figures 11-14 show some examples of the animation in action and I've included a short movie in with the project download if you'd like to see the sand in motion.

Overall, I'm quite pleased with how this worked. It's an astonishingly simple app for a grand effect. I had no idea that simulating sand falling could be handled with so little math and such a basic algorithm. The way the sand tumbles down into little mountains looks amazingly realistic to me, and I do find watching the sand fall entertaining and soothing.

Of course, there are many things I didn't try to implement in this project. Maybe you'd like to expand upon this and add other features, like the ability to drag and drop in various obstacles, move them around, and even draw new ones.

A challenging feature would be to let the user rotate the sand display to any angle (like my circular-shaped toy in real life). Or, if you really wanted to get ambitious, add currents or wind to make the sand swirl. Both of those are way above my pay grade, but I know many of you are programming geniuses and might find it amusing. Share your creations with everyone and have fun!



# New to Xojo?

## Check out *xDev Magazine's* "Welcome to Xojo" Bundle!

### ***Bundle includes:***

- Year 21 of the magazine in printed book format (500+ pages thick)
- Free shipping (up to a \$30 value if you're not in the U.S.)
- Year 22 of the magazine in digital format (PDF)
- A one-year digital (PDF) subscription to *xDev* (beginning with issue 22.1)



That's a total of 12 issues of the magazine—two years worth of great learning!

Bought individually these items could cost as much as \$130, but with this special bundle you can save up to 24%: the *Welcome to Xojo* bundle is **just \$99** with *free shipping* of the book!



<http://www.xdevmag.com/welcome2xojobundle.shtml>