Diffusion-Limited Aggregation

- Start with an immobile seed in a 2D, square grid.
- A walker is then launched from a random position far away and is allowed to diffuse by moving one grid space in a random direction each time step.
- If the walker touches the seed, it is immobilized instantly and becomes part of the aggregate.
- Similar walkers are launched one-by-one and each of them stops upon hitting the cluster.
- Try imagining what the result of this process....
2D Diffusion-Limited Aggregation

By Mark Stock

3D Diffusion-Limited Aggregation

By Mark Stock
By Simon Chorley
Pyrolusite: Manganese Dioxide, MnO$_2$

Photo: wanderflechten of Flickr.com under Creative Commons license

Manganese Dioxide Dendrites on Limestone

Photo: Professor George R. Rossman, Dept Geology, Caltech

Photograph by Mark A. Wilson (Department of Geology, The College of Wooster).
Project 2 Milestone 1: DLA2: (1 of 2)

- Given DLA.java (and Picture.java version 2012.10.24 or newer) on the class website.

- Create DLA2.java that has all the features of DLA.java plus each crystal pixel must be colored one of eight different colors which is determined by the number of its eight neighbors that are crystal pixels.

- For example, the ? pixel has 4 neighbors.

When the ? pixel crystalizes, each of its neighboring pixels will have an additional neighbor. Therefore, when a pixel crystalizes, you need to not only set its color but also change the color of each of its neighbors.
Your DLA2.java must have a total of at least 10 different colors:
- Background (empty pixel).
- Moving Particle color.
- Eight different crystal colors.

Choose a set of colors that look good together. Please step outside of the Crayola 8-pack

You might use different hues and different brightnesses or different brightnesses of the same hue or some other scheme.

DLA2: Grading Rubric

+5 With no more than a few coloring mistakes, when a new pixel crystalizes, its color is determined by its number of neighbors.
+5 With no coloring mistakes, when a new pixel crystalizes, its color is determined by its number of neighbors.
+5 With no more than a few coloring mistakes, the color of each crystal pixel in the image is determined by its number of neighbors.
+5 With no coloring mistakes, the color of each crystal pixel in the image is determined by its number of neighbors.
+5 With 10,000 particles in a 600×600 pixel draw area, the ESCP-110 lab computers can update 1000 frames in no more than 1 minute.

Either change the particle recycling to create partials in a random location on any of the 4 edges or modify the random walk to have a preferred direction with probabilities that generate more fern-like structures.

-1 through -10 for not following the CS-152 coding standard.
DLA2: Extra Credit

+5 Maintain the eight colors for different neighbor counts while adding a gradual shading in brightness or saturation that is determined by the frame count on which the pixel crystalized. For example, hue could be used to show neighbor counts while darker colors of the same hue could indicate "older" pixels.

+15 Rewrite DLA.java to operate on a hex grid. Of course, the pixels will still be square. However, if the neighbor relationships are worked out on a hex grid then larger structures will manifest hexagonal structure. This will allow modeling of snowflake growth.

+30 Do it in 3D using Java OpenGL (JOGL)

DLA2: Path to Success

- Test as you create your program.
- Test with a small window (≤ 25×25) and a small number of particles (≤ 10).
- Whenever a particle crystalizes:
  - Print its index in the parallel arrays (dotX[] and dotY[]), print its coordinates and its number of neighbors.
  - Use the saveImage() method of the Picture class to save a copy of the current off-screen buffer.
  - The saveImage() method of Picture opens a dialog box for the user to select a file name. If you want to call saveImage() on a series many frames, this can get annoying. Overload saveImage() with saveImage(String filename) then generate file names with the frame number (i.e. pic1.png, pic2.png, .... )
saveImage Method of Picture Class

```java
import javax.swing.JFileChooser
import java.io.File;
import javax.imageio.ImageIO

1) public void saveImage() {
2) {
3) JFileChooser fileChooser = new JFileChooser();
4) int value = fileChooser.showSaveDialog(null);
5) if (value != JFileChooser.APPROVE_OPTION) return;
6) 
7) File inputFile = fileChooser.getSelectedFile();
8) String path = inputFile.getAbsolutePath();
9) if (path.endsWith(".png") == false) 
10) { path = path + ".png"; 
11) }
12) 
13) File myFile = new File(path);
14) try 
15) { ImageIO.write(userImage, "png", myFile); 
16) }
17) catch (Exception e) { e.printStackTrace();}
18) }
```

Project 2 Milestone 2: DLA with Sliders

Two additional source files have been provided on the class website:

- **DLA_Sliders.java**: A modification of **DLA.java**
  1) Does NOT contain main.
  2) Has public method for setting the starting crystal(s) and particles.
  3) Has public method for starting/stopping the timer.

- **DLA_CtrlFrame.java**
  1) Does contain main that creates both a **DLA_CtrlFrame** and a **DLA_Sliders**.
  2) Has a slider and two buttons that interact with the **DLA_Sliders** class.
DLA with Sliders: Requirements (1 of 3)

[-5 if not]: Must meet all requirements of DLA2.

[-5 if not]: Must have two windows: one that displays the graphics, and one that contains the GUI controls. The windows must:
- Fit in a 1024×768 pixel area.
- Not overlap when first displayed.
- Cause the program to exit when either is closed.

[-5 if not]: Must implement a button that initializes the model.

[-5 if not]: Must implement a toggle button that toggles the model between pause and running.

[-up to 10 if not]: Must follow CS-152 coding standard.

[+2]: The state of the toggle button must always match the state of the simulation.

DLA with Sliders: Requirements (2 of 3)

[+5 each]: Must implement two sliders that only have an effect when "reset" is pressed:
- Sets number of initial seed crystals (from 1 to 15 with a default of 1).
- Sets number of actively moving particles (from 1 to 25,000 with a default of 10,000).

[+5 each]: Must implement two sliders that effect the model while the timer is running:

- Sets probability of particle crystallization on contact with an edge (from 0% to 100% with a default of 100%).
- Sets probability of particle crystallization on contact with a corner (from 0% to 100% with a default of 0%).
DLA with Sliders: Requirements (3 of 3)

[+4]: In the given version, goofy things happen if the model is started before the reset button is clicked. You must prevent this. One way would be to disable the pause button until the start button is pressed. Another way would be to initialize the model with the default values when the program starts.

[+4]: When the user clicks reset, the model must also pause.

There is no specific requirements on the naming or number of classes used. Of course, general program style requirements do apply to both the naming and number of classes.

There is no requirement to use any of the code I have provided. You may use it, change it or ignore it as you see fit. Of course, if you change the Picture class, then you must remember to submit your version (.java file) with your project submission.

Crystallization Probabilities

In this project, the probabilities of crystallization are cumulative.

Let the probability of edge crystallization be 75% and the probability of corner crystallization be 60%.

A particle at C has 3 chances of crystallizing: 1 edge, 2 corners.

The probability of crystalizing at C is 100% minus the probability of all 3 "rolls" failing:

\[
= 1.0 - (1.0 - 0.75)(1.0 - 0.6)(1.0 - 0.6) \\
= 1.0 - 0.04 \\
= 0.96
\]

\[\text{if } \text{(rand.nextDouble()} < 0.96)\]

How could rand.nextInt(100)<96 cause a bug?
Probabilities: Some Guiding Principles

- If you calculate a probability greater than 1.0, then you have done something wrong.
- If you calculate a probability less than 0.0, then you have done something wrong.
- Something that has a finite, non-certain probability has a probability that is >0 and <1.0 (NOT 0, infinitesimal, nor 1).
- The probability of rolling a 1 on a six-sided die is \( \frac{1}{6} \).
- If a six-sided die is rolled 7 times, then the probability that at least one of the rolls will be a 1 MUST be:
  1) Greater than \( \frac{1}{6} \) (0.1666)
  2) Well above 0.5 (very likely)
  3) Less than 1.0 (non-certain)

DLA with Sliders: Empty Space Extra Credit

+15: With the spawn points being on the edges, as the crystal gets near the edges, the growth pattern is greatly altered. Fix this by embedding the visible screen space within a significantly larger, empty, virtual space.

Spawn particles on the outer edge of this virtual space.

Make the virtual space large enough so that the edges have no obvious effect on the growth pattern.

The trick is to do this without failing the frame rate requirement of milestone 1.
+10: Eliminate the edges by interpreting the grid as topologically equivalent to a torus. Do this by making the top row adjacent to the bottom row and the left edge adjacent the right edge.

Since the screen edges are no longer special places, spawn new particles with equal probability on any non-crystalized pixel.