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

3D Diffusion-Limited Aggregation

By Mark Stock

By Simon Chorley
Pyrolusite: Manganese Dioxide, MnO₂

Photo: wanderflechten of Flickr.com under Creative Commons license

Manganese Dioxide Dendrites on Limestone

Photograph by Mark A. Wilson (Department of Geology, The College of Wooster).
Lab 4 DLA: (1 of 2)

- Given Picture.java on the class website.
- Create DLA_yourName.java that implements Diffusion-Limited Aggregation on a 2D, 800x800 grid of pixel.
- Initialize the grid with one seed crystal near the grid’s center.
- Each timestep, spawn 100 moving particles along the bottom (south) edge of the grid.
- Each timestep, every non-crystalized particle moves, with equal probability, north, south, east or west by one pixel.
- If a particle moves out of the window, then it gets removed from the particle list (or recycle it by resetting its location, to one of the 100 newly spawned particles).
Lab 4 DLA: (2 of 2)

- If a particle moves adjacent (in the 4 directions) to a crystalized particle, then the moving particle crystalizes (remove it from the particle list and it never moves again).
- Moving particles may pass through each other.
- The background color must be black.
- Moving particles must be white.
- Crystalized particles must be colored something other than black or white. The color you choose must be a function of time so that the first crystalized particles are a different color than the later particles. There must be at least 5 different levels of color.
- Choose colors and age breaks that look good together.

Extra Credit: DLA Hex: +10 points

+10 DLA on a hexagonal grid. Of course, the pixels will still be square. However, if the neighbor relationships are worked out on a hex grid then Aggregated crystals will manifest a hexagonal structure.

Hex grid mapping to Java Rectangular 2D array.
Extra Credit DLA 3D: +10 points

Model on a 3D, 800x800x15 grid.

Draw everything on a 2D board were all z values are projected down onto the x-y plane.

Particles must not pass through each other, but particles with the same x,y and different z may occupy the same pixel.

Draw particles using transparency so that when the thickness is one, the particle looks is clearly, but faintly and when the thickness is 15, the pixel is totally opaque.

Extra Credit Probabilistic DLA: +5 points

In the given model, there is a 100% chance that a particle 4-adjacent to a crystalized particle will crystalize.

Implement this, but also implement a user option so that there is one probability (<100%) of crystalizing if north, south, east or west of a particle and a lesser (but >0%) probability of crystalizing if adjacent to a corner.

You may make these probabilities functions of time.

Find probability functions that produce beautiful results. This will take some experimentation.
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 array coordinates.
- 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`, ....)

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

class Picture
{
    public void saveImage()
    {
        JFileChooser fileChooser = new JFileChooser();
        int value = fileChooser.showSaveDialog(null);
        if (value != JFileChooser.APPROVE_OPTION) return;

        File inputFile = fileChooser.getSelectedFile();
        String path = inputFile.getAbsolutePath();
        if (path.endsWith(".png") == false)
        { path = path + ".png";
        }

        File myFile = new File(path);
        try
        { ImageIO.write(userImage, "png", myFile);
        }
        catch (Exception e){ e.printStackTrace();}
    }
}
```
Three Methods for Draw Update

**Method 1:** Maintain one array all moving particles and one of all crystalized particles. Every update, *after* all particles have moved and spawned, fill the frame with the background color, then draw all crystal particles and all moving particles.

**Method 2:** Maintain only the array all moving particle. Every update, *before* particles have moved, draw each moving particle in the background color. Then, *after* particles have moved, draw each moving particle in the particle color.

**Method 3:** Maintain the array all moving particle and a background image. Every update, draw each *newly* crystalized particle on the background image. Then, *after* particles have moved, draw the full background image in the window. Then, draw each moving particle.