Diffusion-limited aggregation

Diffusion-limited aggregation is the process whereby particles undergoing a random walk due to Brownian motion cluster together to form aggregates of such particles. [Wikipedia]

Instructions

https://github.com/fcooper8472/cpp-cbl-2019

The purpose of this project is to build a computational simulation of diffusion-limited aggregation (DLA). This document will walk you through the task step-by-step, and then give you suggestions for additional work if you finish the main tasks.

Today you will:

1. Work in groups of 6 (or 5)
2. Write C++ code as instructed below
3. Make a PowerPoint (or similar) presentation
    a. Include your names on the first slide
    b. Show the progress your team has made
    c. Indicate which parts you found hard
    d. Show examples of any outputs you generated

Tomorrow morning you will:

1. Present your work in your group (max 10 minutes)
2. Listen to the other presentations
3. Ask questions of the other groups (max 5 minutes)

The DLA algorithm

1. Start with a single particle in the middle of a regular square grid
2. Add a new particle randomly at the edge of the grid
3. Let the new particle move randomly until it is directly neighbouring the existing particle
4. Repeat with many new particles, moving them randomly until they join the aggregate

The C++ task

1. Create a class called `DiffusionSimulator`
    a. Add three private members to the class:
        - `int mNumRows;`
        - `int mNumCols;`
        - `std::vector<std::vector<int>> mGrid;`
2. Write a main function, and check that you can create an instance of `DiffusionSimulator`:
    a. `DiffusionSimulator sim;`
3. Add a constructor with signature `DiffusionSimulator(int num_rows, int num_cols)`

   a. Your constructor should appropriately resize the grid, and set all the values in the grid to zero
   b. Remember that to resize a `std::vector<std::vector<int>>` you first resize the outer vector to hold `num_rows` elements, and then each of the inner `std::vector<int>` should be resided to hold `num_cols` integers

4. Check that you can now construct your object:
   a. `DiffusionSimulator sim(25, 15);`
5. Write a `public` member function with signature `void print()` that will print `mGrid` to the console
   a. Check that `sim.print()` prints a grid of all zeros, with 25 rows and 15 columns
6. Write a `public` member function with signature `void set(int row, int col)` that will set a specific element of the grid to the value 1
   a. Check that `sim.set(12, 7)` sets the middle element of the grid to 1
7. Write a `private` member function with signature `bool check(int row, int col)` to check whether a specific element of the grid is equal to 1
   a. If `(row, col)` is outside the grid (e.g. `row` or `col` is -1), then `check` should return `false`
8. Write a `private` member function with signature `bool check_neighbours(int row, int col)` that will check if any *neighbouring* elements of the grid are equal to 1
   a. This function should call `check` a number of times
   b. You can decide which directions count as being neighbours
9. Write a `private` member function with signature `void update_location(int& xPos, int& yPos)` that will randomly move `xPos` and `yPos` to a valid neighbouring square (not outside the grid)
   a. You will need to decide how to handle the case where the random move would put the point outside the grid
10. Write a `public` member function with signature `void simulate()` that will start a particle at a random location on the edge of the grid and keep on randomly updating its location until it is next to another particle already in the grid
    a. This function should call `check_neighbours`, `update_location`, and `set`
11. Write a for loop in the main function that will call `sim.simulate()` a large number of times
    a. Then call `sim.print()` and see what pattern is generated

Additional ideas

1. Write a function that prints the grid to a file, and use other software to visualise the aggregate?
2. What if you start with more than one particle in the grid?
3. What if you bias the direction of the random motion?
4. How long does it take each particle to aggregate?
   a. Plot a graph of number of steps against particle number
5. Anything else you can think of..?