# Project 01: Falling Sand

**Assigned:** Monday, April 22, 2019
**Due:** Wednesday, May 8 at 11:30pm

## 1 Acknowledgements

This is a heavily modified version of an assignment originally written by Dave Feinberg of Columbus Academy and presented at the Nifty Assignments session at SIGCSE 2017. Information about the original version can be found at `http://nifty.stanford.edu/2017/feinberg-falling-sand/`.

## 2 Objective

The guessing game that we wrote in Project 00 was at least slightly entertaining, I hope, but probably not something that you spent more than 5 minutes running after you were sure that it worked correctly. It also required you to do exactly what I wanted, exactly as I wanted it done.

This project will have a lot more "replay value" and also be much more open-ended, allowing you to engage your creativity.

As part of this project we will become much more comfortable working with two-dimensional arrays and writing classes with instance variables and instance methods. We will also get more practice reading and writing files.

We are going to build a falling sand game. Because it has no score, goals, or opponent, it should probably be called a toy or simulation rather than a game, but the name is already in popular use. A falling sand game allows the user to "draw" different types of particles – metal, sand, water, and others – on a canvas, similarly to how you might draw with a graphics program. Unlike a basic painting program, however, these particles will move and interact with each other.

## 3 Pair Programming

As with the first project, you may complete the project with a partner as long as you use the pair programming technique throughout the process. If you would like to work with a partner but do not have anyone in mind, I recommend posting about your interest in Piazza / answering someone else's post on Piazza to find a partner.

## 4 Setup

Continue using the same Eclipse workspace, but create another project named `Project01`.

Download `Project01Files.zip` from D2L and unzip its contents into the `src` directory of your project. (Then refresh Eclipse so that it sees they are there.) You will find three files:

- `SandDisplay` is a class that creates the graphical window we can play the game in. You will not need to modify this class at all. In fact, you do not even need to read it, though you may if you would like to.

- `FallingSandProgram` is a class that contains a `main` method that runs the program. You might want to make minor modifications to this file to adjust the size or speed of simulation, but are not required to do anything with it.
- `Simulation` is the class that stores information about where each particle is and moves them around. You will be making very many changes to this class.

Find the AutoLab page for `Project 01: Falling Sand`. As in the previous labs and projects, it would be a good idea to submit your file after each question so that if there are problems with your work you can fix them before moving on. Because you are only modifying the `Simulation` class, that is the file that you will submit to AutoLab.

## 5 Assignment

Please complete each of the steps below.

1. (0 points) Run the program. Unlike all of our other programs that interacted through the console, it will open a graphical window. You will not be able to do much yet, but you should see a big black rectangle where you will soon be able to draw and four buttons named `Save`, `Load`, `Empty`, and `Metal` that do not yet do anything.

   Then start reading through the `Simulation` class. You should see the following:

   - A constant that stores an array of strings. These are the names of the buttons that you saw (without `Save` and `Load`). Later, we will add more strings to the array.
   - A list of constants for each type of particle. We need these to each be associated with the matching index into the array above for that type of particle.
   - A list of constants for each color that will be used for drawing particles. You can read about the `Color` class by searching the web for its documentation.
   - No fields yet, though you will be adding some soon.
   - A constructor that does not yet do anything, though you will be changing that soon.
   - Lots of methods for which I have written the header (so that the other classes will compile) but left the body empty (so that they do not yet work correctly). You will be writing the bodies of these methods, one at a time.

2. (15 points) Get the base game working, which will allow you to paint with Metal and Empty (essentially an eraser). To do so, complete the following steps. You should be able to compile after each one, but the program will not work correctly until all of them have been completed.

   (a) Add a field to the `Simulation` class that stores a two-dimensional array of integers. You may name this field anything you would like, but I am going to refer to it as the grid throughout the rest of these instructions. The grid is going to store one of the constants above (`EMPTY` or `METAL`) for every location in the picture.

   (b) In the `Simulation` constructor, initialize your field by creating a new two-dimensional array. Use the constructor's parameters to decide how large it will be.

(c) Fill in the bodies of the `getHeight` and `getWidth` methods. Note that you no longer have access to the constructor's parameters, since you are no longer in the constructor. Do not create any more fields to help with this – you should be able to get the answers from the grid.

(d) Fill in the bodies of the `fillLocation` and `getParticleType` methods. The former changes what is stored at a location in the grid, while the latter returns what is stored at a location in the grid.

(e) Fill in the body of the `updateDisplay` method. The job of this method is to iterate over the entire grid, calling `setColor` on the `SandDisplay` with the coordinates of that grid location with the color that matches the particle type there (`EMPTY_COLOR` for `EMPTY` or `METAL_COLOR` for `METAL`).

(f) Create a method named `toString` that returns a `String` with the following contents:

- A first line containing the height, a space, and the width.
- A line for each row of the grid, containing each element in that row separated by spaces (with an extra space after the last one).

When you run the game now, you should be able to press the `Metal` button and then click / drag on the picture to draw silver metal particles. You should also be able to press the `Empty` button and click / drag on the picture to erase metal particles.

Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on. There's no easy way for AutoLab to test `updateDisplay`, but just make sure that you can see the particles on the picture when you run the program.

3. (5 points) Add sand to the game. To do so, complete the following steps:

(a) Add `"Sand"` to the end of the `NAMES` array.

(b) Below the `EMPTY` and `METAL` constants, create one named `SAND` that has a value of 2.

(c) Below the `EMPTY_COLOR` and `METAL_COLOR` constants, create one named `SAND_COLOR` that has a value that represents a yellow-brown color. `https://www.rapidtables.com/web/color/RGB_Color.html` can help you find a color you like and tell you what parameters to use to get it.

(d) Update the `updateDisplay` method to set the `SAND_COLOR` as well.

When you run the game now, you should see an extra button named `Sand`, and you should be able to draw with it, just like the others.

Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on.

4. (10 points) Animate sand. While metal just stays wherever we put it, we do not want sand to. When it does not have anything solid beneath it, sand falls downward (following the law of gravity). If it falls off the bottom of the picture, it should disappear. We will need to go through several steps in order to make this happen:

(a) Create a new method named `swapLocations`. It should have four parameters: the row and column of one particle and the row and column of a second particle. It does not return anything. It swaps the locations of those two particles. So if the first location contained sand and the second metal, when the method finishes the first would contain metal and the second sand.

(b) Create a new method named `updateSand`. It should have two parameters: the row and column of a sand particle. If that sand particle is already in the lowest row, it should be replaced by empty. Otherwise, if the space below the sand particle is empty, you should swap its location with the location below it. If that particle has sand or metal below it, you should do nothing.

(c) Add another field to the class, which will store a `Random` object. In the constructor, initialize that field to a new `Random`.

(d) Fill in the body of the `doOneUpdate` method. This method should choose a random location in the picture to update. If what is in that location is empty or metal, do nothing. But if it is sand, call the `updateSand` method for that location.

Now when you paint with sand you should see that it falls toward, and eventually off, the bottom of the screen. You can prevent it from falling off the bottom by painting metal somewhere, then letting the sand fall onto it. The slider at the bottom of the screen determines how fast the sand falls. If you find that even the slowest setting is not slow enough for you, or even the fastest setting is not fast enough for you, adjust the `speedFactor` variable in the `main` method (of `FallingSandProgram`) until it feels right.

Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on.

5. (15 points) Allow the user to paint with water, by repeating all of the steps taken for sand. (Add it to the array of names, create the `WATER` and `WATER_COLOR` constants, create a `updateWater` method, etc.) Follow the same naming convention. Use a blue color. Water behaves in the following way:

If it can fall downward, it does. If there is anything other than empty below it, it will instead randomly pick left or right. (You might think of this as rolling a 2-sided die.) If the chosen direction would move it off the screen, it does so. Otherwise, if the chosen direction would move it into an empty space, swap it with that space. If the chosen direction would move it to a filled space, do nothing.

Now that we have added water, we should notice that the behavior of sand no longer makes sense: sand should not sit on top of water – if sinks! Modify `updateSand` so that sand swaps itself with both empty and water that is underneath it.

You should be able to paint with water now and see that it fills a volume just like you would expect a liquid to. (Try making a metal bowl, then dumping water into one side of it.) You should also see that sand sinks through water.

Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on.

6. (10 points) Allow the user to save their pictures and load them. There are two steps:
   (a) Fill in the body of the `save` method, which will be called whenever the user presses the `Save` button and enters a file name. We want to write the result of `toString` to the file.
   (b) Fill in the body of the `load` method, which will be called whenever the user presses the `Load` button and enters a file name. We need to read the contents of the file and use them to replace the grid with a new one.

   You should now be able to save a picture, make some changes, and then load the original picture again. If you implemented saving and loading correctly, you could even save a picture of a certain size, then change the number of rows or columns (see the `main` method), run the program again, and load the old file that has a different size. You may need to manually resize the window to get the entire image visible after doing so.

   Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on.

7. (20 points) Allow the user to paint with acid, by repeating all of the steps taken for sand and water. Follow the same naming convention. Use a pink color. Acid behaves in the following way:

   - 80% of the time, it does the same thing water would do.
   - 15% of the time, it randomly chooses one of the four cardinal directions around it, and if that location is filled with water it swaps positions with the water. If that location is not filled with water, it does nothing.
   - 5% of the time, if it has metal or sand below it, it dissolves that metal or sand (converting it to empty) while staying where it is at. If it does not have metal or sand below it, it does nothing.

   The behaviors of empty, metal, and water can remain unchanged, but we need to make a futher change to sand: it should sink in acid the same way that it sinks in water.

   Test that acid fills a container like water does, mixes with water in a solution, and eventually burns its way through sand and metal.

   Submit to AutoLab, so that if there are any problems with this part you can find and fix them before moving on.

8. (15 points) [not autograded] Design three other particles that have their own unique behaviors. Use a color for each that will stand out against the existing particle types. For each one, include a detailed comment above its `update` method that explains exactly what it does, and why. Think about how it would interact with all of the existing particle types, and adjust each of their `update` methods appropriately.

   I would love it if you came up with your own original ideas, but here are some to get you thinking:

   - Smoke or other gas particles might float upward toward the sky.

- Base particles might act like acid particles except that they dissolve sideways rather than down, and annihilate acid particles they come into contact with.
- Lava particles might turn into metal if they are surrounded on all sides by water.
- Algae particles might eat water particles to multiply.
- Black hole particles might pull everything nearby toward them, destroying things as they crunch toward the center.
- Slippery sand might be willing to move sideways, but only if there are at least three consecutive empty / water / acid spots under where it would move to.

Ideas that are both exceptionally creative and difficult to implement correctly may, at the instructor's discretion, earn extra credit to be added to your lowest lab grade.

9. (10 points) AutoLab will now automatically check for Javadoc comments and good style. (It can't find all types of bad style, though, so I will still be checking some things.) Ensure that you are making it happy and that you think you will make me happy as well.

## 6 Post-Assignment Thoughts

Let your friends and family members play your game! Save pictures of the most interesting scenarios you can set up with your particles, so that you can show them off. Think about the value of simulations like this not just for fun but also for modeling and better understanding scientific phenomena.

## 7 Submitting Your Work

Make sure that you have submitted your final version of `Simulation.java` to AutoLab, and that (unless you are satisfied with a lower score) you have 85/85 autograded points.

You do not need to submit your work through D2L – I will look at whatever your last AutoLab submission is.