Creating a Falling Sand Simulation

Andy Hayes and Carl Liu

Macalester College – St. Paul, MN

Introduction

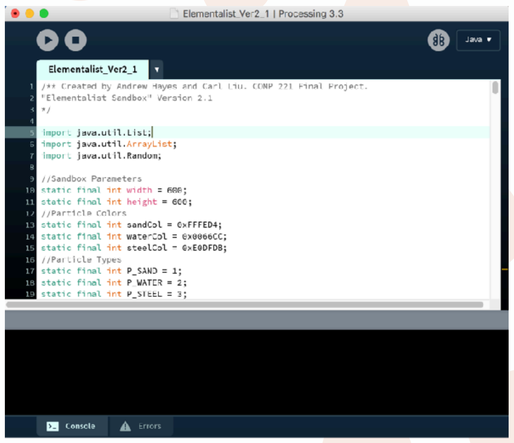
The goal of our project was to create and implement our own algorithms for a 2 dimensional falling sand style simulation program. In this program users would be able to create and drop multiple different materials, such as sand or water, into the environment and watch how they interact.

Initial Goals for the Project

- We want to create a functional program with realistic interactions between different materials.
- Ideally we would like to implement many different materials including sand, water, oil, cement, and fire
- Realistic interactions between different elements
- The final product should look cool and be fun to play with for at least a few minutes.

Software Used

- We coded this project completely in Java
- We used the Processing 3 IDE.



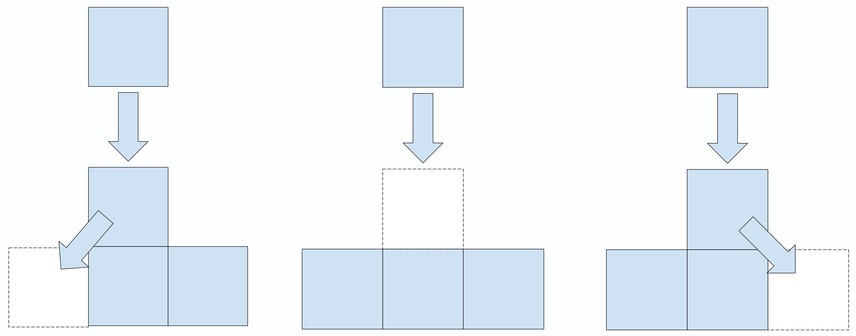
Implementation

- Our environment is a 600 x 600 grid of pixels.
- We used a 2D array to represent the grid of pixels and would assign each position an integer if the pixel was occupied by a particle.
- Our algorithms would iterate over the 2D array and update the position of every particle every frame

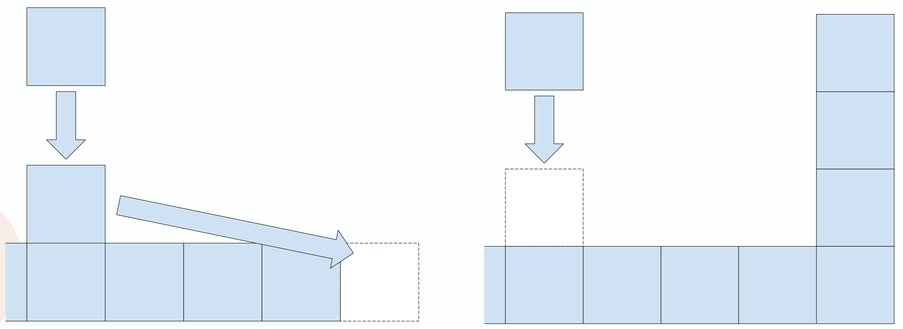
Implementation cont.

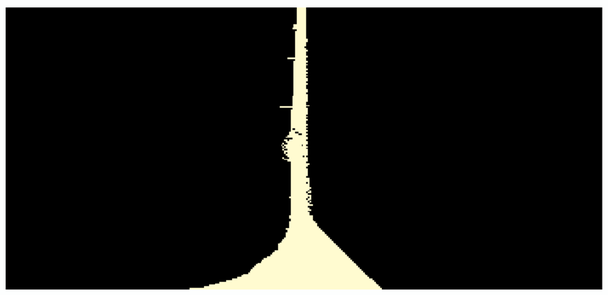
- We needed to create two algorithms to define the particle movement. One for solids and one for liquids because they will react differently

Solid Materials



Liquid Materials





Pseudocode

```
Algorithm updateSolid(Particle p, xcoord, ycoord, grid[][])
1.  Particle q = grid[x][y+1]  // get the positions of the spaces below the particle
2.  Particle l = grid[x-1][y+1]
3.  Particle r = grid[x+1][y+1]
4.  if q == null:     //check if something is there
5.      p.moveTo(x, y+1, grid)
6.  elif l == null and r != null:
7.      p.moveTo(x-1, y+1, grid)
8.  elif r == null and l != null:
9.      p.moveTo(x+1, y+1, grid)
```

```
Algorithm updateLiquid(Particle p, xcoord, ycoord, grid[][])
1.  Particle q = grid[x][y+1]  // get the positions of the spaces below the particle
2.  Particle l = grid[x-1][y+1]
3.  Particle r = grid[x+1][y+1]
4.  if q == null:     //check if something is there
5.      p.moveTo(x, y+1, grid)
6.  elif l == null and r != null:
7.      p.moveTo(x-1, y+1, grid)
8.  elif r == null and l != null:
9.      p.moveTo(x+1, y+1, grid)
10. else:
11.     p.moveTo(x + random(-1,1), y, grid)
```

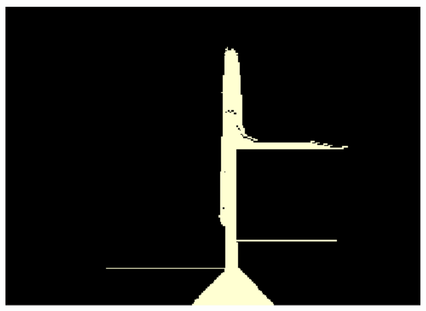
Time Complexity

These algorithms run over the entire grid[][] of points. In the updateSolid algorithm, the basic operation is checking if the position below the particle is null or not, this is a constant time operation so for N points in the grid our time complexity is O(N).

The updateLiquid algorithm is very similar to the update solid, except when the algorithm for solids would leave the particle where it sits, the liquid algorithm makes the particle move one direction or the other, this results in the particles eventually leveling out as water should. The time complexity of this algorithm is also O(N).

Difficulties

One difficulty we faced was successfully implementing the algorithm for updating liquid particles. At first we were attempting a solution that involved helper collision detection algorithms, but we were able to slightly modify our algorithm for solids to adapt it for liquids. Another difficulty we faced and failed to fix is that sometimes when many particles are being operated on at one time sometimes they do not react exactly as they should and long horizontal lines of particles are constructed as they fall. A picture can be seen below



Acknowledgments

We would like to acknowledge Shilad Sen for teaching us the skills necessary to do this project.

Bibliography

How To Make a "Falling Sand" Style Water Simulation. (n.d.). Retrieved April 30, 2017, from http://w-shadow.com/blog/2009/09/29/falling-sand-style-water-simulation/