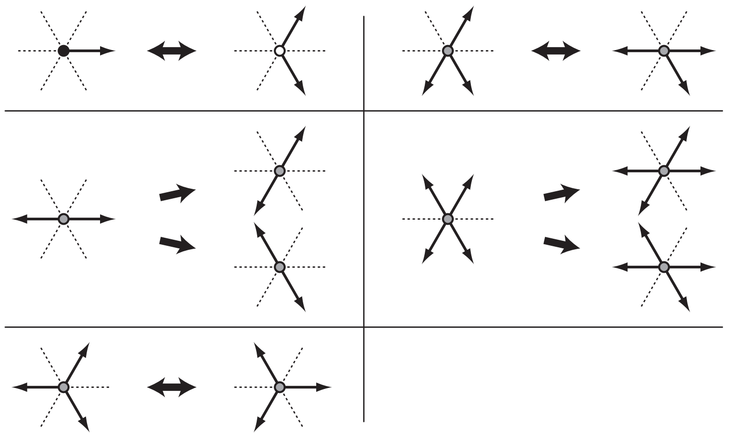
Physics 750: Assignment 1

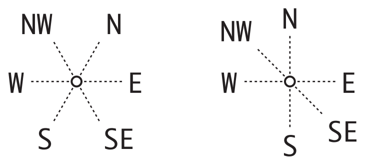
due Tuesday, September 19, 2017

In this assignment, you'll be exploring the FHP lattice gas introduced by Frisch, Hasslacher, and Pomeau [Phys. Rev. Lett. **56**, 1505 (1986)]. The automaton consists of a regular triangular array of sites holding identical particles of unit mass that are either stationary or moving with one of six discrete velocities of equal magnitude directed along the lattice links. No more than one particle per site can exist in any of these seven states. Particle updates are carried out in two passes. The first is a propagation step in which each non-stationary particle is made to move along its velocity vector to an adjacent site. The second is a collision step in which the following net-momentum-preserving reconfigurations are carried out.

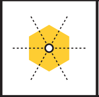
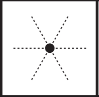
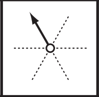
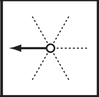
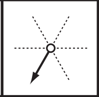
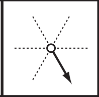
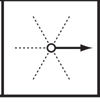
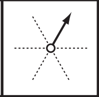


In the figure above, black sites are occupied by a stationary particle, and white sites are unoccupied. The occupancy of the grey sites is unspecified and may be zero or one. When there are two possible reconfiguration pathways—as is the case for the two- and four-velocity collisions—one of the branches is chosen randomly with probability 1/2.

To start, download and unpack the `assignment1.tgz` archive from the class website. The `assignment1` directory contains a program file `lattice_gas.cpp` that (partially) implements an extension of the FHP model that allows for reflection off fixed obstacles. In the code, the triangular lattice is sheared so that it can be represented as a square lattice (which is itself stored as a one-dimentional C array in row-major order). In this geometry, two of the nearest neighbours to each site are connected diagonally across a square plaquet. The six nearest-neighbour directions are described by an enumerated type `ver_t` and named for the corresponding compass points:



The program makes use of a compressed storage format. The state of each lattice site is stored in a single byte, with each of the eight bits flagging the presence of a velocity, stationary particle, or obstacle.

`0x80`	`0x40`	`0x20`	`0x10`	`0x08`	`0x04`	`0x02`	`0x01`
							

Issuing the commands

```
$ make
g++ -o lattice_gas lattice_gas.cpp -O2 -ansi -pedantic -Wall
g++ -o lattice_gas_openGL lattice_gas.cpp -O2 -ansi -pedantic -Wall ...
$ ./lattice_gas_openGL
Usage: lattice_gas_openGL (setup=0,1,..,4) (0 < concentration < 1)
```

from inside the `Assignment1` directory will build and execute the program. Note that two executables are produced—one that outputs to the terminal only, and another that animates the resulting hydrodynamics in a separate window using the openGL library. Both programs require that two command line parameters be specified by the user. The first selects one of five simulation modes: (`0`) uniform gas; (`1`) circular dilution wavefront; (`2`) linear dilution wavefront; (`3`) enclosed container with unequal pressures inside and outside; (`4`) fluid flow past an elliptical barrier. The second parameter controls the number density of the fluid.

1. (4 points) The lattice connectivity is defined through a collection of functions named `indexN`, `indexE`, . . . , `indexNW`. Implement these functions, making sure that the boundary conditions are properly defined for the triangular lattice. (Remember that, with regard to the internal representation, the lattice has been sheared so that it coincides with an orthogonal grid. This means that wrap-around in one of the directions is no longer trivial.) Simulate a uniform gas at extremely low densities and verify by visual inspection that each particle *correctly* wraps around the edges.

2. (4 points) The function `report` writes the total particle number and the total x- and y-directed momenta to the terminal after each time step. The particle number at each site, computed with `occupancy`, is accumulated in an integer variable `n`. The momenta, stored as double-precision floating-point variables `px` and `py`, are incremented using the `inc_momentum` function. Write the body of the `occupancy` and `inc_momentum` functions. (Remember that the `occupancy` function counts all particles, both moving and stationary!) Verify that `n`, `px`, and `py` are conserved quantities.

```
$ ./lattice_gas 0 0.3
         860178           293       -289.252
         860178           293       -289.252
         860178           293       -289.252
         860178           293       -289.252
         860178           293       -289.252
         860178           293       -289.252
         860178           293       -289.252
```

3. (6 points) Extend the `collision` function to account for four-velocity collisions. Check that `n`, `px`, and `py` are still conserved.

4. (3 points) Run `lattice_gas` in modes `1` and `2` for various values of the concentration. How does the propagation of the disturbances depend on the particle density? How does the behaviour change when you turn off all collisions? Give me a brief physical explanation for your observations.

```
void animate(int)
{
   propagate();
   //collision();
   if (is_flowing) flow();

   report();

   glutPostRedisplay();
   glutTimerFunc(delay,animate,0);
}
```

5. (5 points) Code up a special case for mode 3 so that the particle number and momenta inside and outside the container are counted separately. Write to the terminal in six-column format.

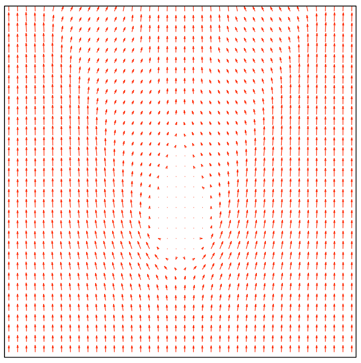
```
cout << setw(15) << n_in << setw(15) << n_out
     << setw(15) << px_in << setw(15) << px_out
     << setw(15) << py_in << setw(15) << py_out << endl;
```

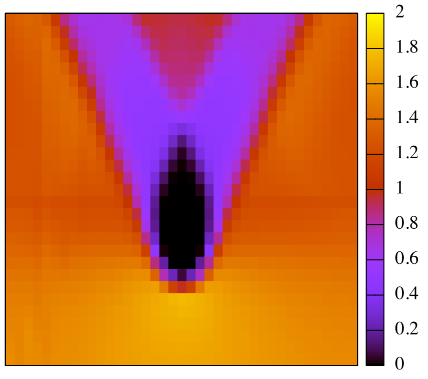
Run the simulation in mode 3 and dump the output to a file:

```
./lattice_gas 3 0.65 > outfile.dat
```

Make a plot of the `n_in` and `n_out` values versus the simulation clock. Do the same for `px` and `py`. Comment on the behaviour of the curves, paying special attention to the oscillations and the decay envelope. Use fits to extract the relevant time constants. Explain the (physical or implementation-dependent) origin of these phenomena.

*6. (7 points) Run the simulation in mode 4 with concentration `0.1`. Roughly determine the time needed for the flowing liquid to reach its steady state. Measure the steady-state velocity and occupation number fields, time-averaged and coarse-grained over several lattice sites. Produce a vector plot of the resulting smoothed velocity field and a density plot of the smoothed occupation number field. (Inspect the files `view4b.gp` and `view4c.gp` in the solutions to Exercise 3 for a `gnuplot` template; and have a look at `mandelbrot-png.cpp` and its `makefile` in Exercise 1 for an example of how to use `pnglib`.)




2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0