Physics 750: Exercise 3

Thursday, August 30, 2017

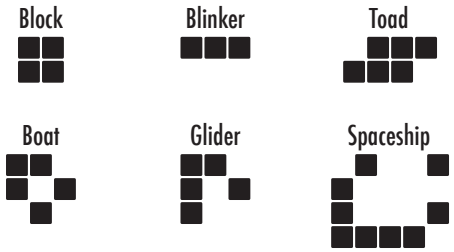
1. Use the `curl` command to download from the class website everything you'll need for this exercise.

```
$ WEBPATH=http://www.phy.olemiss.edu/~kbeach/
$ curl $WEBPATH/courses/fall2017/phys750/src/exercise3.tgz -O
$ tar xzf exercise3.tgz
$ cd exercise3
```

2. The "Game of Life"[†] is a famous *cellular automata*[‡] system introduced by John Conway in 1970. It consists of a square lattice of cells that are either alive or dead. The status of each cell evolves under the influence of its eight neighbours. The lattice is updated simultaneously according to the following rules.

   - if two of its neighbours are alive, the status of a cell remains unchanged
   - if three of its neighbour are alive, the cell will be alive at the next time step
   - otherwise, the cell will be dead at the next time step

   This simple system gives rise to surprisingly complex behaviour. The configurations of live cells shown below evolve in a regular way.



   A C++ implementation is provided in the `life.cpp` file.

```
$ make life
g++ -o life life.cpp -O2 -Wall -ansi -pedantic -lglut
$ ./life
```

   When you run `life`, the initial state of the system is established by the function `initialize_grid`, which sets all the cells to dead and then creates two blinkers and a glider.

```
create_blinker(10,10);
create_blinker(25,25);
create_NW_glider(40,30);
```

   You'll see that the blinkers flip between vertical and horizontal orientation and that the glider moves uniformly in the north-west direction. The speed of the animation can be controlled by changing the integer `delay`, which stores the frame delay in milliseconds. To stop the program, type `q` or return to the originating terminal window and hit `[ctrl-c]`.

[†] https://en.wikipedia.org/wiki/Conway's_Game_of_Life
[‡] https://en.wikipedia.org/wiki/Cellular_automaton

(a) Modify the initialization routine so that a glider is set on a collision course with a blinker.

```
create_NW_glider(50,50);
create_blinker(30,70);
```

Recompile using `make` and run the program again.

(b) Write functions that create blocks, boats, toads, and spaceships. How do these behave?

(c) Modify the rules of life so that

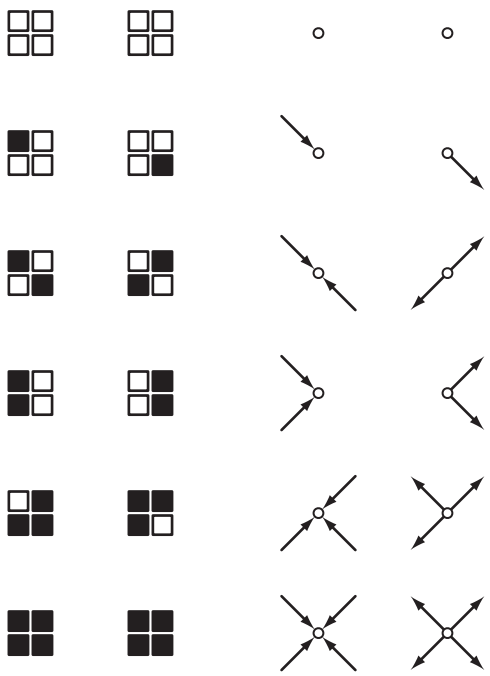
- if n of its neighbours are alive, the status of a cell remains unchanged
- if $n + 1$ of its neighbour are alive, the cell will be alive at the next time step
- otherwise, the cell will be dead at the next time step

The conventional game corresponds to $n = 2$. See what happens for $n = 1, 3, 4$.

(d) Explore what happens if you eliminate the periodic boundary conditions.

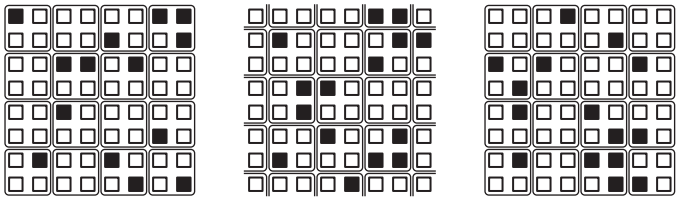
3. In class, we introduced a momentum-conserving model of identical particles moving on the square lattice.[§] These particles have unit velocities restricted to one of the four lattice directions, and no two particles on the same site can have the same velocity value. This lattice gas model can be recast as a cellular automata on a square lattice dual to the original (rotated by $\pi/4$, with the cells centred on the links). Each automata cell has two states (on or off, alive or dead) corresponding to whether or not a velocity vector passes through the cell. The directionality of the vector is encoded by assuming that the vectors alternate at every time step between being outward- and inward-pointing on 4-cell square clusters.

   The update rules include free motion of the particles and all allowed multi-particle collisions.



[§] https://en.wikipedia.org/wiki/HPP_model

The updates are carried out by sweeping over a grid of 4-cell squares, alternately offset by (0,0) and (1,1).



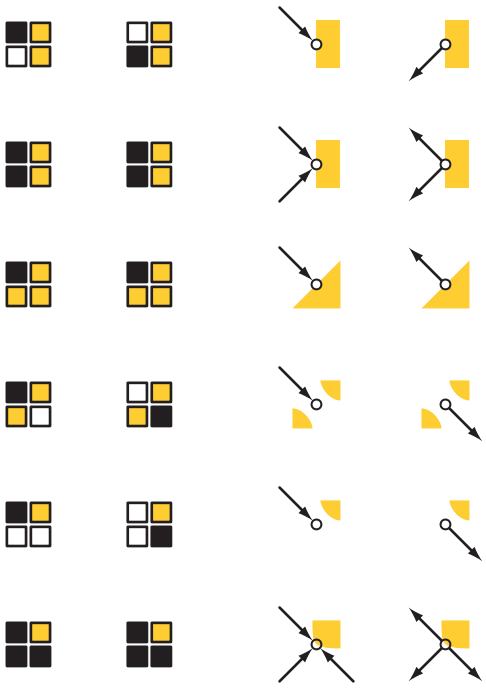
(a) Observe the evolution of the automata system from an initial configuration in which the active cells are distributed randomly with a uniform concentration.

```
$ make hpp
g++ -o hpp hpp.cpp -O2 -Wall -ansi -pedantic -lglut
$ ./hpp
Usage: hpp (setup=0,1,2) (0 < concentration < 1)
$ ./hpp 0 0.5
```

(b) Write additional code (in the function `verify`) that checks whether the two orthogonal momenta (p_x and p_y along the directions of the original lattice) and the particle number are conserved. Have the function send a suitable warning message to the standard error stream (`cerr`) if they are not.

(c) Run `hpp` with `setup=1`, which initializes the system so that a low density hole sits in the lower left quadrant. Try various values of the concentration, including values very close to one. What is unusual about the hole's motion? Why can you observe quasi-periodicity at high concentrations? What can you say about the rotational symmetries of this fluid? Will coarse graining help?

4. Extend the automata system so that each cell has three possible states: off, on, and obstacle—corresponding to `fluid[i][j]` equal to `0`, `1`, and `2`. Rewrite the `update` function so that it handles all possible specular reflections.



(a) Run `hpp` for various concentrations with `setup=2`. Verify that the number of gas particles and obstacles are each constant and that the obstacles remain fixed in place.

(b) For a system of size L, the time scale for particles to propagate across the system is roughly L. Measure the local particle density, coarse-grained in 2×2 blocks and averages over the time steps $50L < t \leq 75L$. Output the data to a file and plot as a two-dimensional false-colour map. (Hint: type "`help pm3d`" in `gnuplot`.)

(c) In the same way, measure the two components of the local velocity and produce a vector plot like the one shown below. (Hint: type "`help vectors`" in `gnuplot`.)