

Physics 420/580: Lab 2

Thursday, September 23, 2010

1. Log in to your account using the userid and password you chose during Lab 1. Your userid is of the form p420u1xx or p580u1xx. If you do not have an account yet, please talk to your TA.
2. Open a terminal window from the menu in the top-left corner of the screen:

Applications ▸ Accessories ▸ Terminal

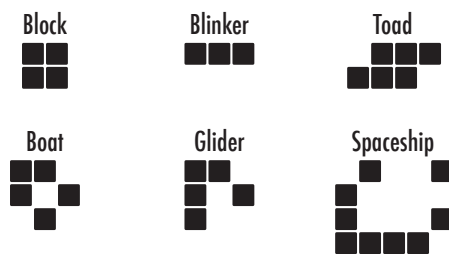
3. Download the Lab 2 instructions and source code from the class website. You can either do this from a web browser or from the terminal using the `curl` command.

```
$ WEBPATH=http://www.ualberta.ca/~kbeach/phys420_580
$ curl $WEBPATH/docs/Lab2.pdf -O
$ curl $WEBPATH/src/Lab2.tar.gz -O
$ tar xzf Lab2.tar.gz
$ cd Lab2
```

The pdf instructions (this document) can be viewed onscreen by typing `evince Lab2.pdf`.

4. 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.



More details can be found at http://en.wikipedia.org/wiki/Conway's_Game_of_Life

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. Hit [ctrl-c] in the terminal to stop the program.

(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

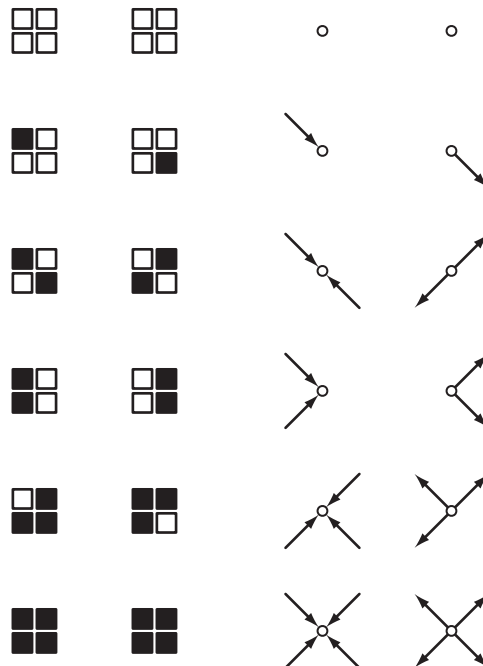
- if n of its neighbours are alive, the status of a cell remains unchanged
- if $n + 1$ of its neighbour are alive, the cell will be alive at the next time step
- otherwise, the cell will be dead at the next time step

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

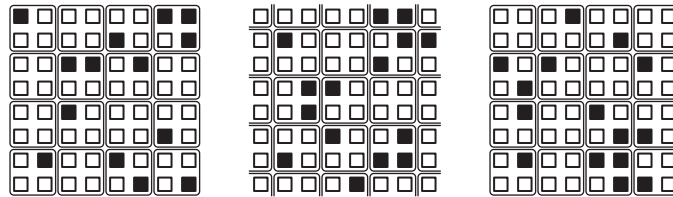
(d) Explore what happens if you eliminate the periodic boundary conditions.

5. 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.



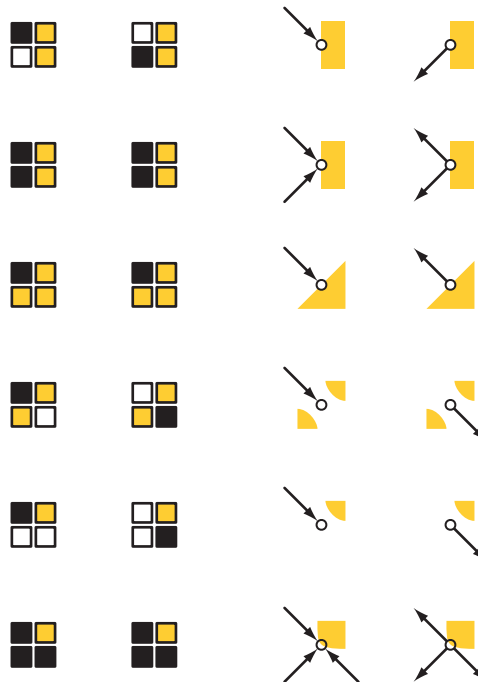
The updates are carried out by sweeping over a grid of 4-cell squares, alternately offset by (0,0) and (1,1).



- (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: lattice_gas (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 if they are not.
- (c) Complete the unwritten part of the `sweep_grid` function that handles the case of open boundary conditions. Run `hpp` for various concentrations with `setup=1`. Show that the total momentum is no longer conserved. Why is this so? What can you say about the rotational symmetries of this fluid?
- *6. Extend the automata system so that each cell has three possible states: on, off, and obstacle. Rewrite the update function so that it handles all possible specular reflections.



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.

7. Before you leave, make sure to close out of your session using [System ▸ Logout](#).