ACCU 2022

CROWD YOUR WAY OUT OF A PAPER BAG

FRANCES BUONTEMPO

Crowd Your Way out of a Paper Bag

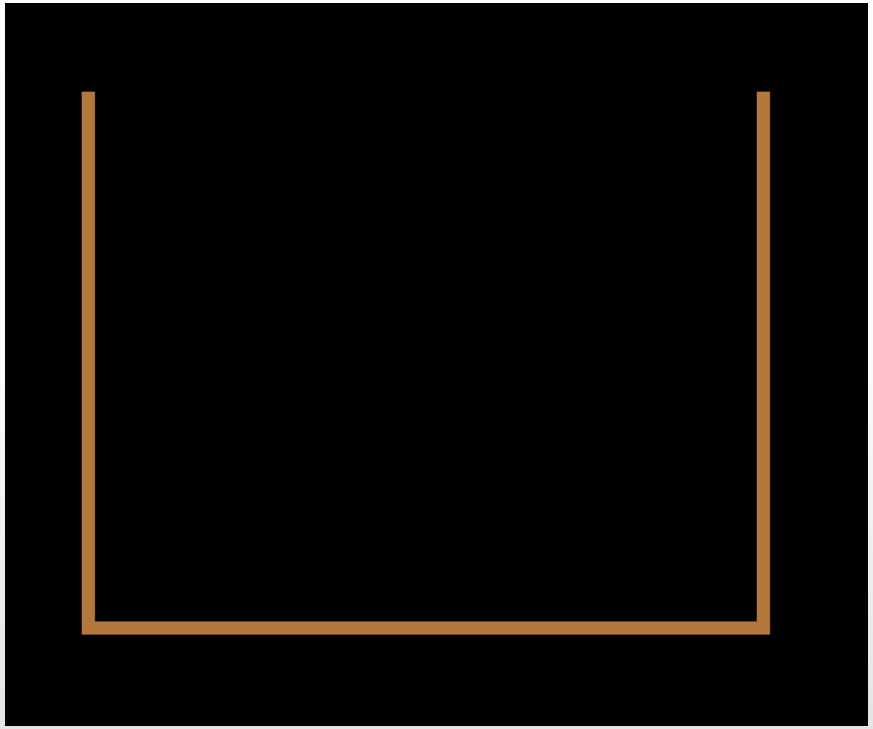
It’s not rocket science!

Frances Buontempo

April 2022

Contents

- Swarms
- Follow the leader
- Multi-lane follow the leader
- Cellular automata
- Agents
- Paper bags



Can you code your way out of a paper bag?

Particle Swarm Optimisation

- Particles have memory, and so does the “swarm”
- Each particle has a velocity in 3 parts
  - current – initialized randomly
  - towards personal best
  - towards swarm’s best
- We need to define “best”
- Take a weighted sum of these
  - With a bit of randomness thrown in
- We need to choose these weights

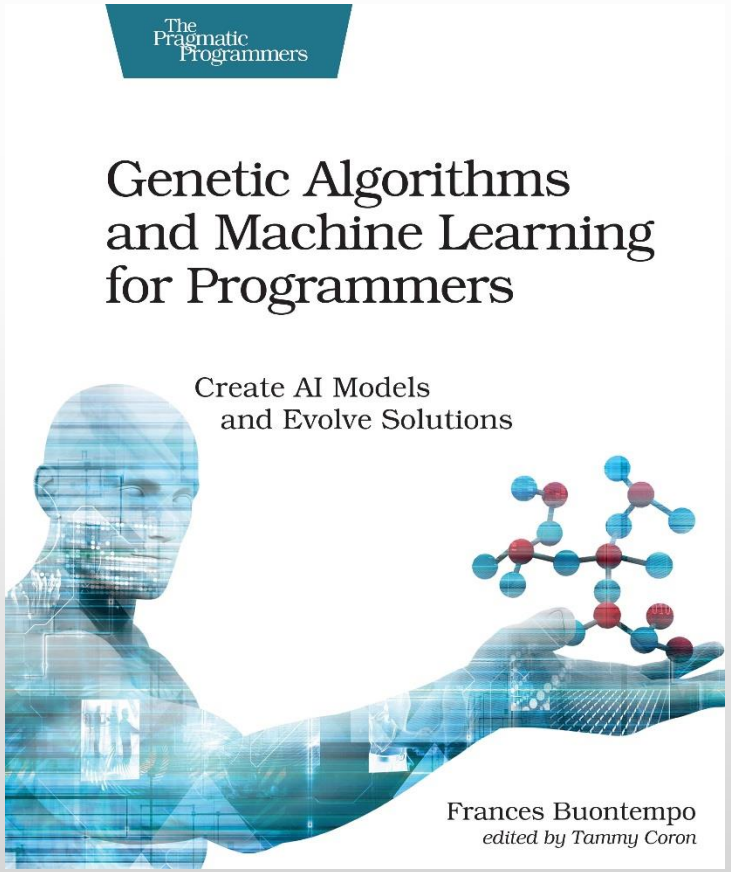
Algorithm

```
Choose n
 Put n particles in random points in the bag
 While some particles are still in the bag
    Update best global position
    Draw particles current positions
    Move particles
        updating each particle's
        current best position
```

Swarms

- Continuous movement
- A personal velocity, memory and “hive mind”
- PSO, https://accu.org/index.php/journals/2023

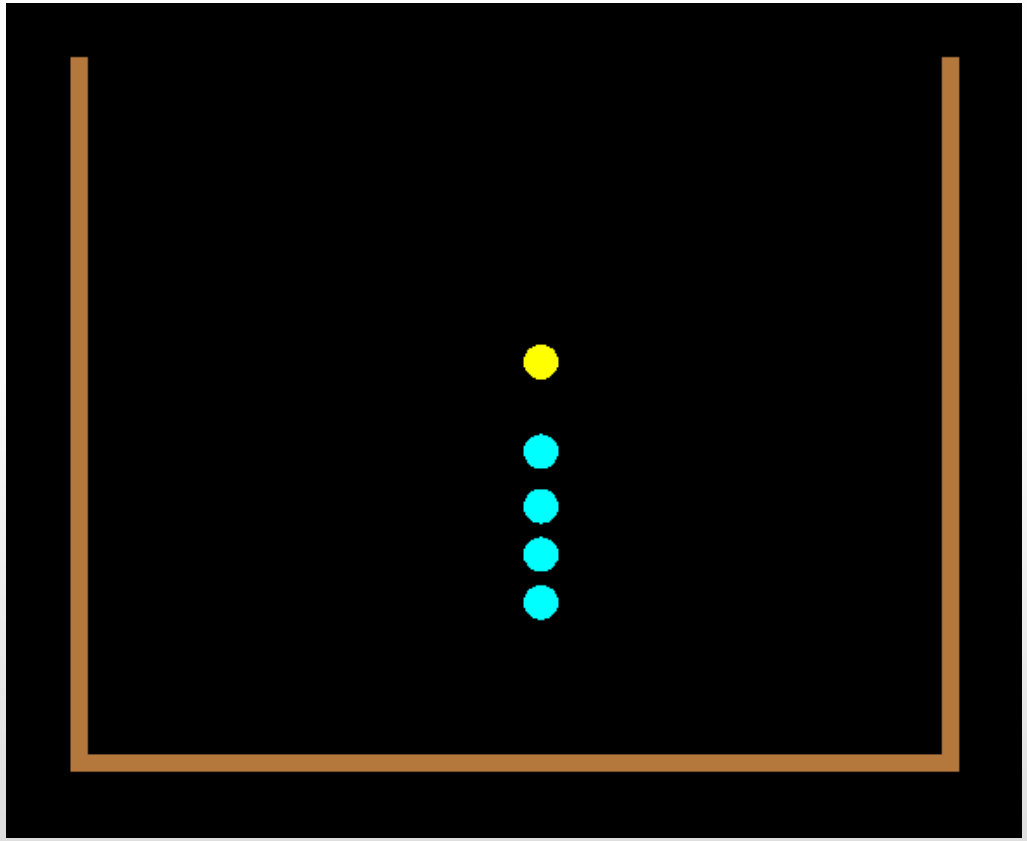
https://pragprog.com/book/fbmach/genetic-algorithms-and-machine-learning-for-programmers



Follow the leader

Imagine a lane of traffic or people or “particles” or blobs

- in an orderly one lane queue
- and move at a constant speed



Blobs in (continuous) space

- No memory
- A preferred velocity / distance to the blob in front
- Called φ, phi, for reasons

```
std::function<float(float)> phi_steady =
    [](float dt) {return dt * 100.0f; };
```

Blobs in space

```
struct Blob {
    float x;
    float y;
    std::function<float(float)> phi;

    void move(float dt){
        y += phi(dt);
    }
};
std::vector<agents::Blob> blobs;
```

What if the leader stops?

- Leader does what they want
- Others pay attention to the blob in front
  - Otherwise they walk through each other
- Just need one small change to φ

Leader does what they want

```
std::function<float(float, float, float)> phi_pause =
 [](float time, float dt, float w) {
        if (time > 3.5f && time < 8.5f)
            return 0.f;
        else
        {
            return dt * 100.0f;
        }
    };
```

Followers have to pay attention

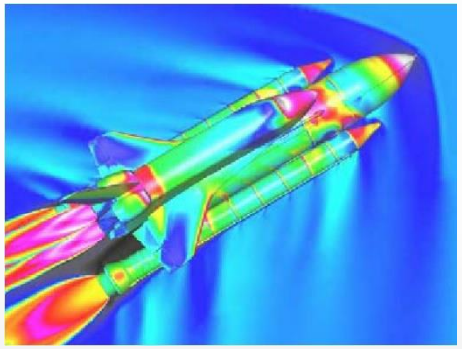
- See how close the next one is and adjust accordingly
  - For example, keep 2 metres apart
  - So we don’t need a collision detection
  - According to maths
- Let’s send w (for distance to next blob) to φ

Social distancing - It's ~~not~~ rocket science

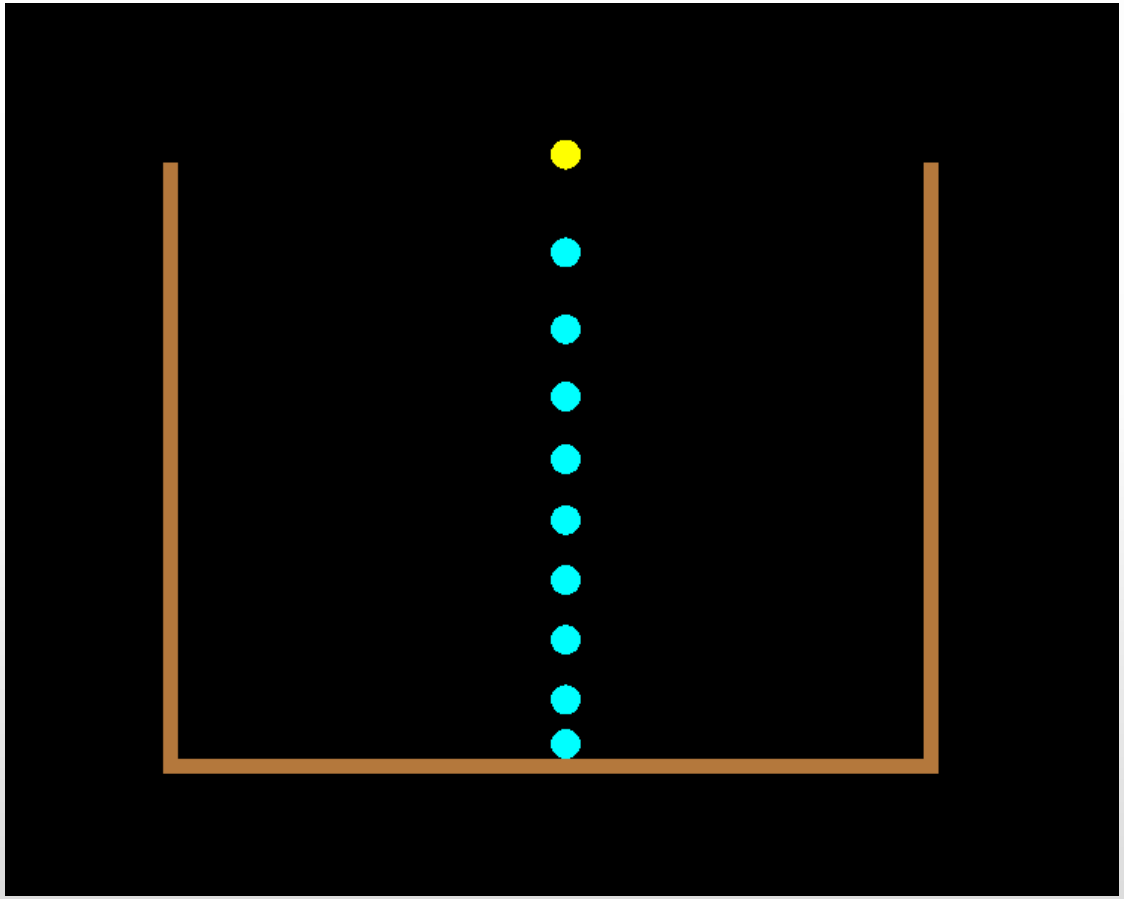
```
std::function<float(float, float, float)> phi_steady =
[](float time, float dt, float w) {
    if (w > 40.f) {
        return dt * 100.f *
               (1.0f - std::exp(-(w - 40.f) / 25.f));
    }
    else
        return 0.f;
};
```

The Transport Equation/Fluid Dynamics

- Microscopic versus Macroscopic
- Discrete: $\frac{dx_i}{dt} = \varphi(x_{i+1} - x_i)$
- Continuous: $\frac{\partial \rho}{\partial t} + \nabla . j = \sigma$
  - ∇ , nabla is divergence (a gradient/rate of change)
  - j is flux (like a flow rate)
  - σ , sigma is for sources and sinks – it's 0 for us – "incompressible"



Has Fran done the demos yet?

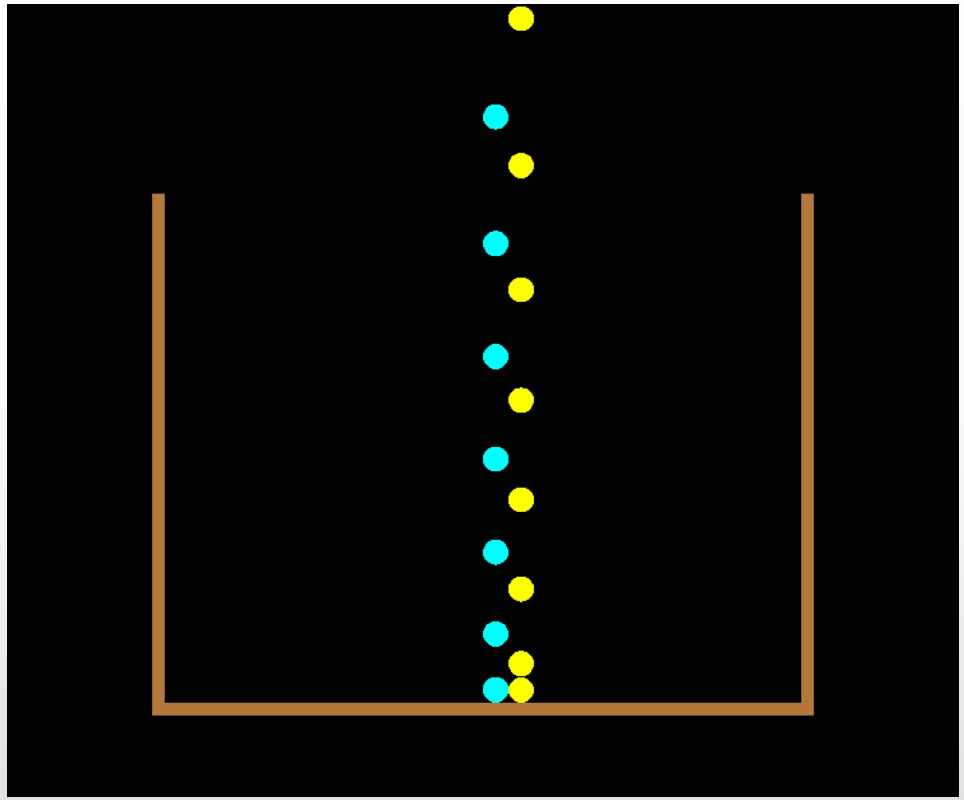


Ghost Jam!

- The “traffic” jam moves upstream
- AKA Stop and Go waves or phantom jams
- Also triggered by lane changing, which we’ll see in next

Multi lane

- Let's have two "lanes"
  - Implemented by two vectors of blobs
- To start thinking about *agency*
  - The blobs only had a preferred velocity
    - Though did slow down to avoid crashing
  - Blue blobs have the same old phi function
  - Yellow blobs might change lane (vector)
    - For no apparent reason



“Traffic models based on cellular automata have high computational efficiency because of their simplicity in describing **unrealistic vehicular behavior** and the versatility of cellular automata to be implemented on parallel processing. On the other hand, the other microscopic traffic models such as car-following models are computationally more expensive, but they have more realistic driver behaviors and detailed vehicle characteristics.”

A multi-lane traffic simulation model via continuous cellular automata

Emanuele Rodaro, Öznur Yeldan

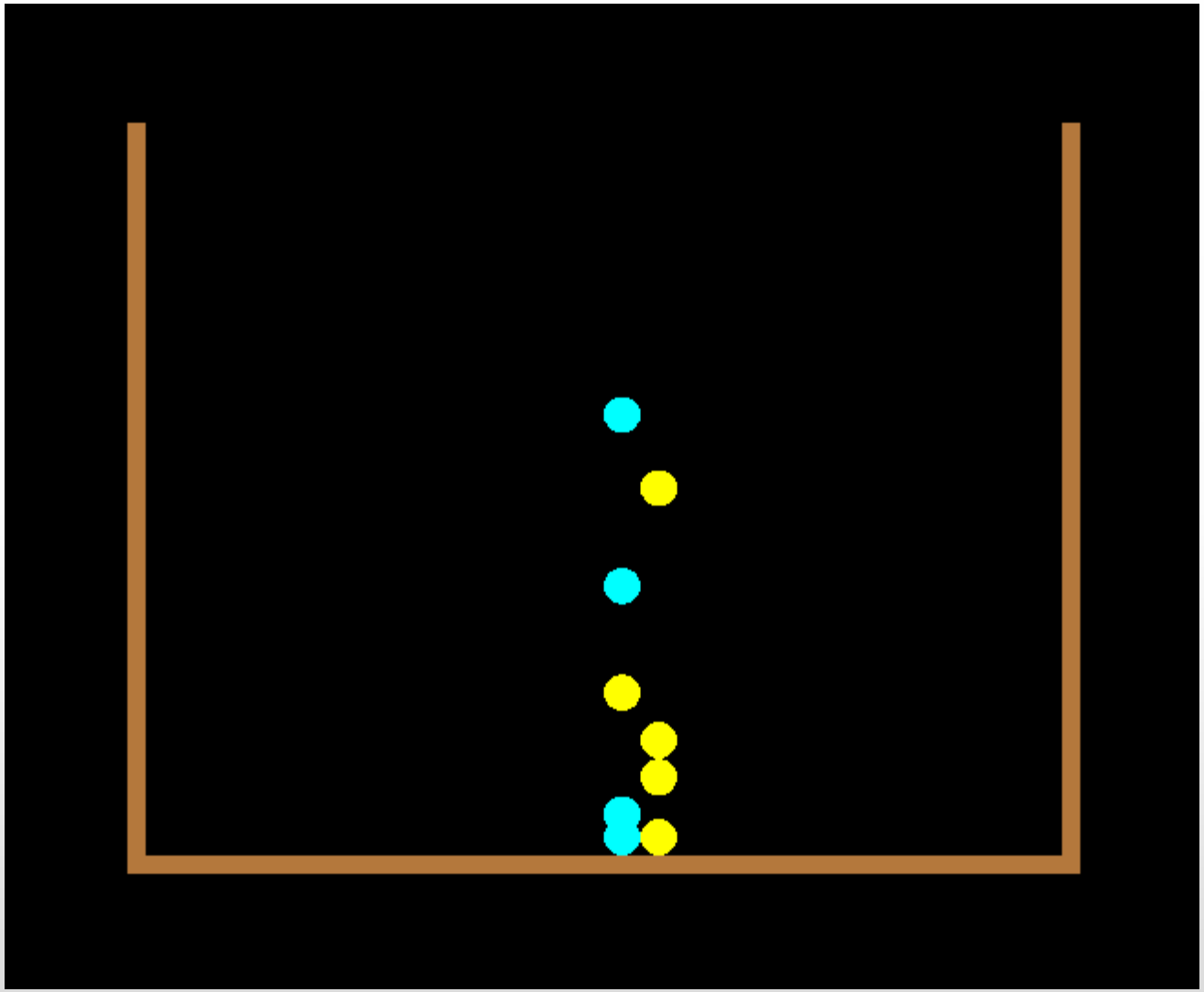
https://arxiv.org/abs/1302.0488

So, let’s do something completely different

Trouble

- The paper cited is **way** more sophisticated than what I did
  - They build “a stochastic cellular automata traffic model in which the space is not coarse-grain but continuous”
- Let’s use two vectors and some blobs will switch between the right or left hand lane

```
void move_lane(std::vector<Blob>& left,
      std::vector<Blob>& right, RNGBase & r);
```



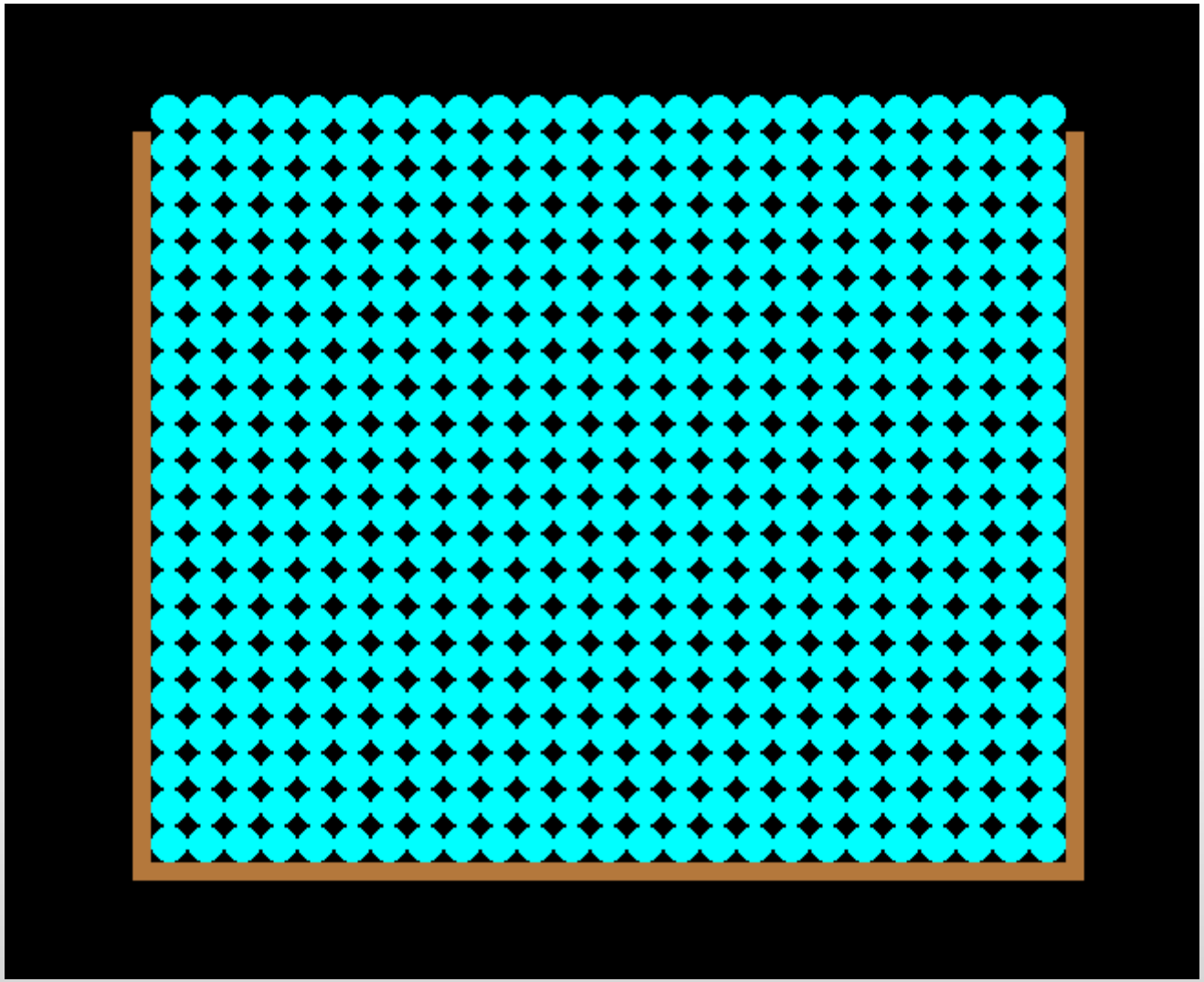
Multi-lane FtL

- Did Fran do any demos?
- AKA Faster is Slower
- Possible extensions:
  - Some tend to move left only (or right only)
  - “Politeness”
  - Make lane changing velocity based
  - Add traffic lights, roundabouts etc
- https://traffic-simulation.de/ is fun

What next?

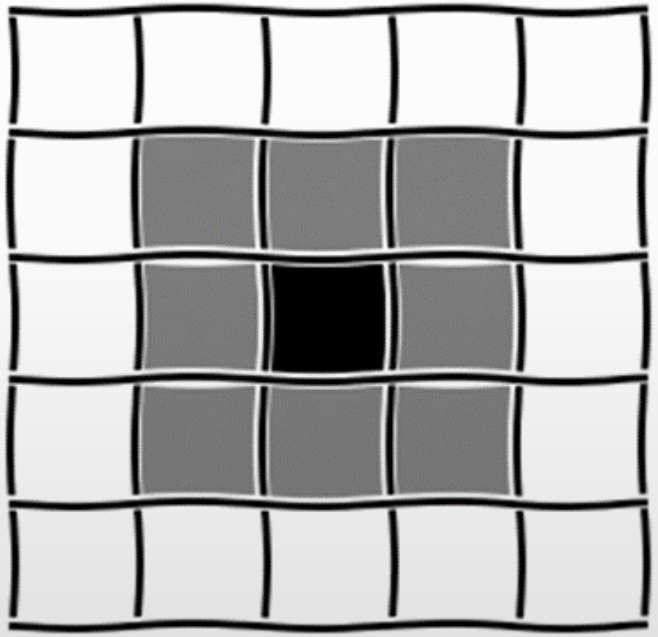
- `std::vector<Blob>`
- `2 std::vector<Blob>`
- `std::vector<std::vector<Blob>>`
- Use the whole paper bag!

Cellular automata

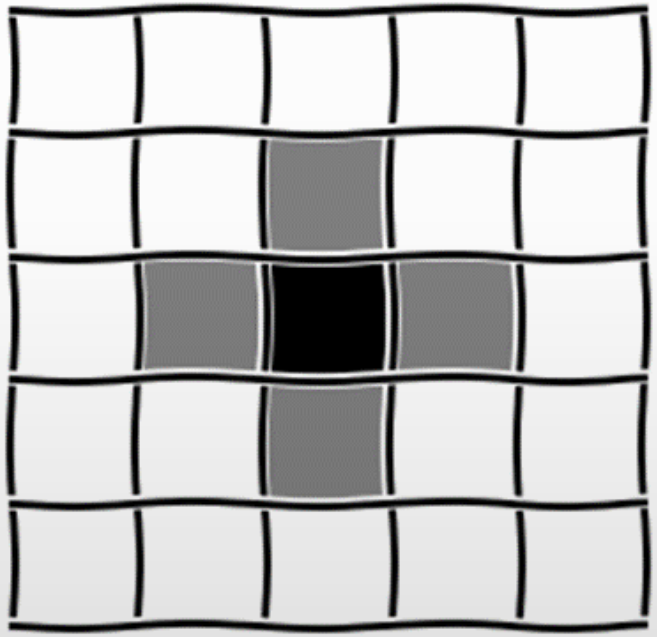


Neighbours

Moore

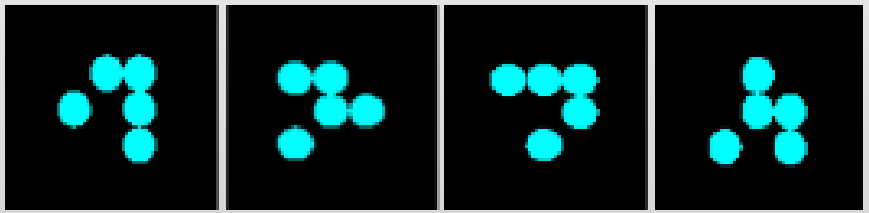


von Neumann

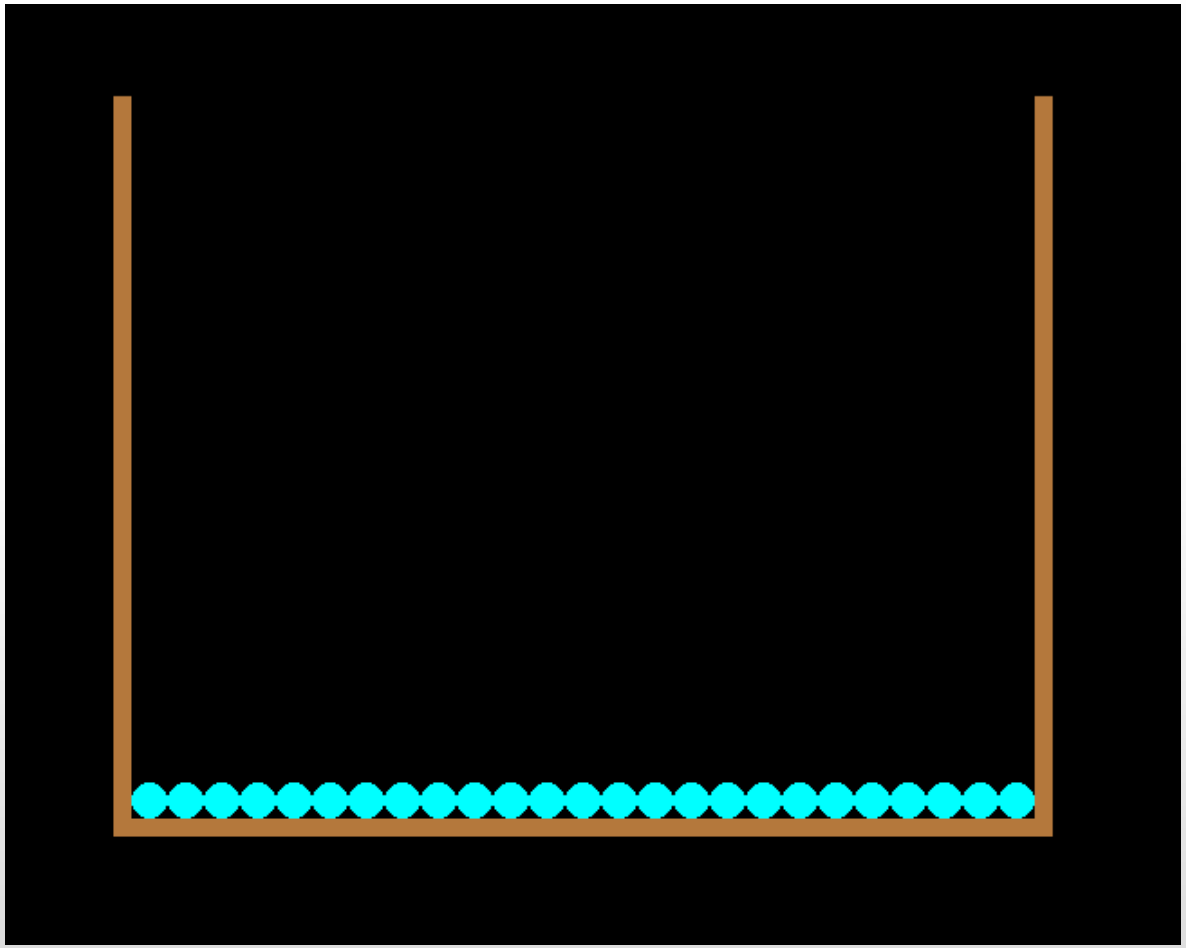


Conway's Game of Life

- A live cell with fewer than two live neighbours dies, as if by underpopulation.
- A live cell with two or three live neighbours lives.
- A live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.



Crowd your way out of a paper bag



Stochastic cellular automata

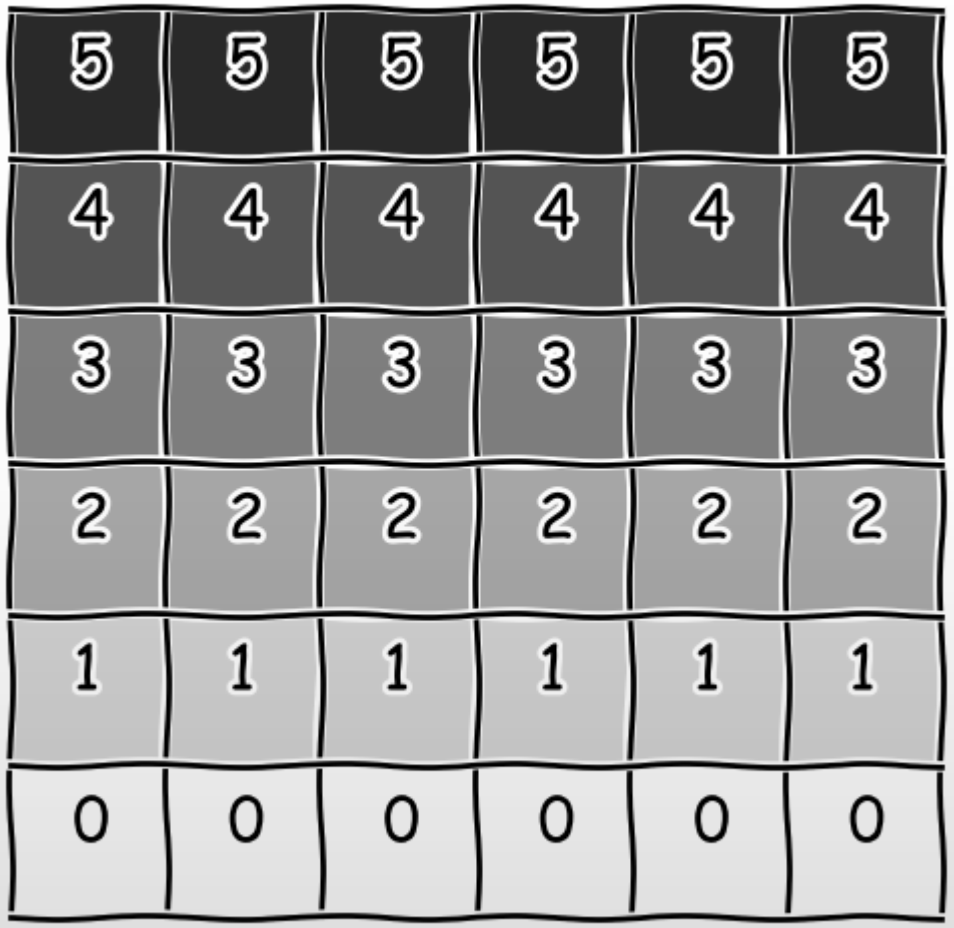
1. Possible moves – von Neuman
   - `std::vector<std::pair<int, int>> moves{ { 0, 0 }, { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };`
   - Also, don't stand on someone else
2. Probabilistic choice
   - Blob will weigh up the options
3. Two update schemes
   - One at a time
   - Or all at once (like the Game of Life)

Possible moves

```
class Grid {
public:
    Grid(size_t x, size_t y); //...
private:
    std::vector<std::vector<Blob>> grid;
};
```

- Allowed?
  - Don’t bust through sides of bag
  - Don’t stand on someone/something else

Door Field



Probabilistic choice

- Static floor field – cheat for now

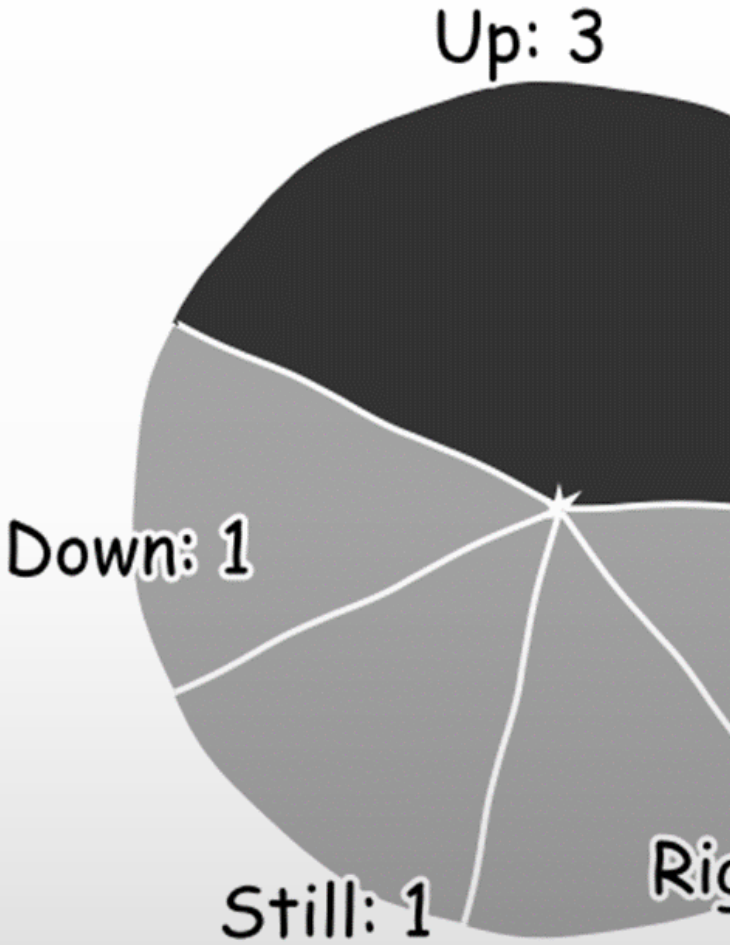
```
int Blob::weighting(int x_move, int y_move) const {
    return (y_move > 0) ? 3 : 1; // 0 if not allowed
}
```

- Then choose:

```
std::discrete_distribution<> d(w.begin(), w.end());
auto index = d(g); // std::mt19937
return moves[index];
```

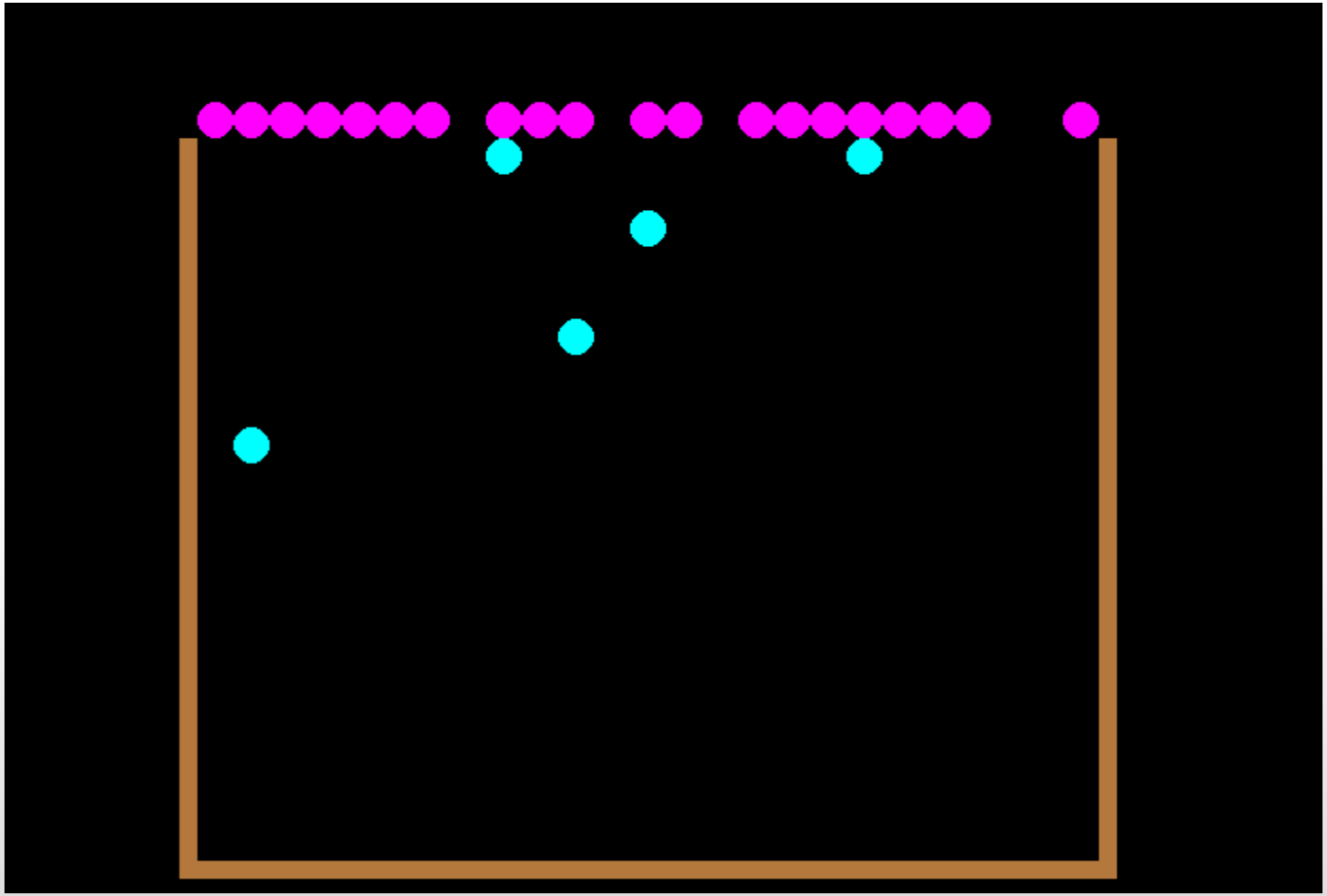
- Dynamic floor fields are possible

Roulette

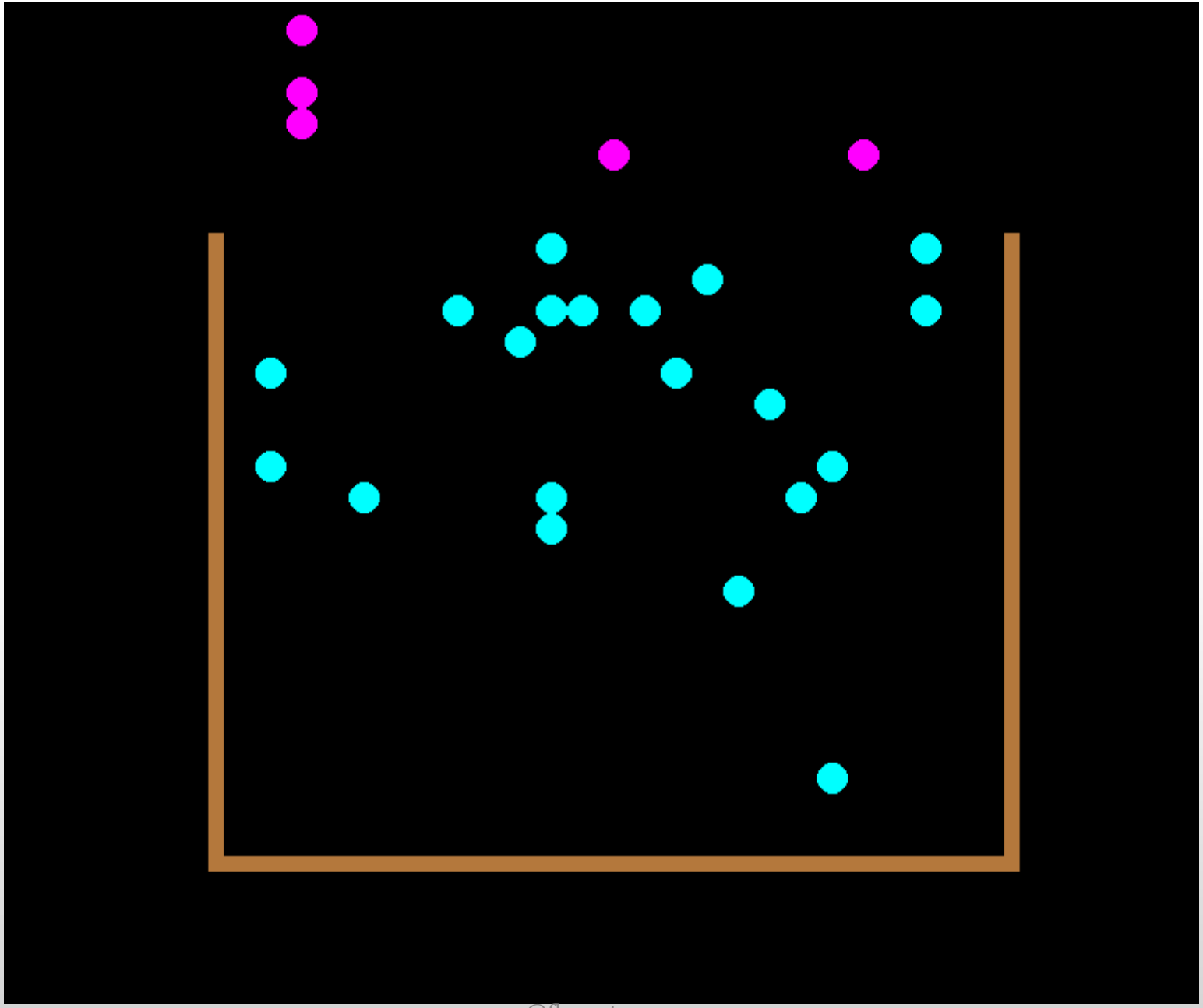


Update scheme

- One at a time
  - None will try to move to the same spot
- You can do everybody moving at once
  - But need a way to detect people going to the same spot
  - and decide what to do in this case
- Recall swarms from before
  - They updated simultaneously
  - (and shared info)
- Note to self - demo



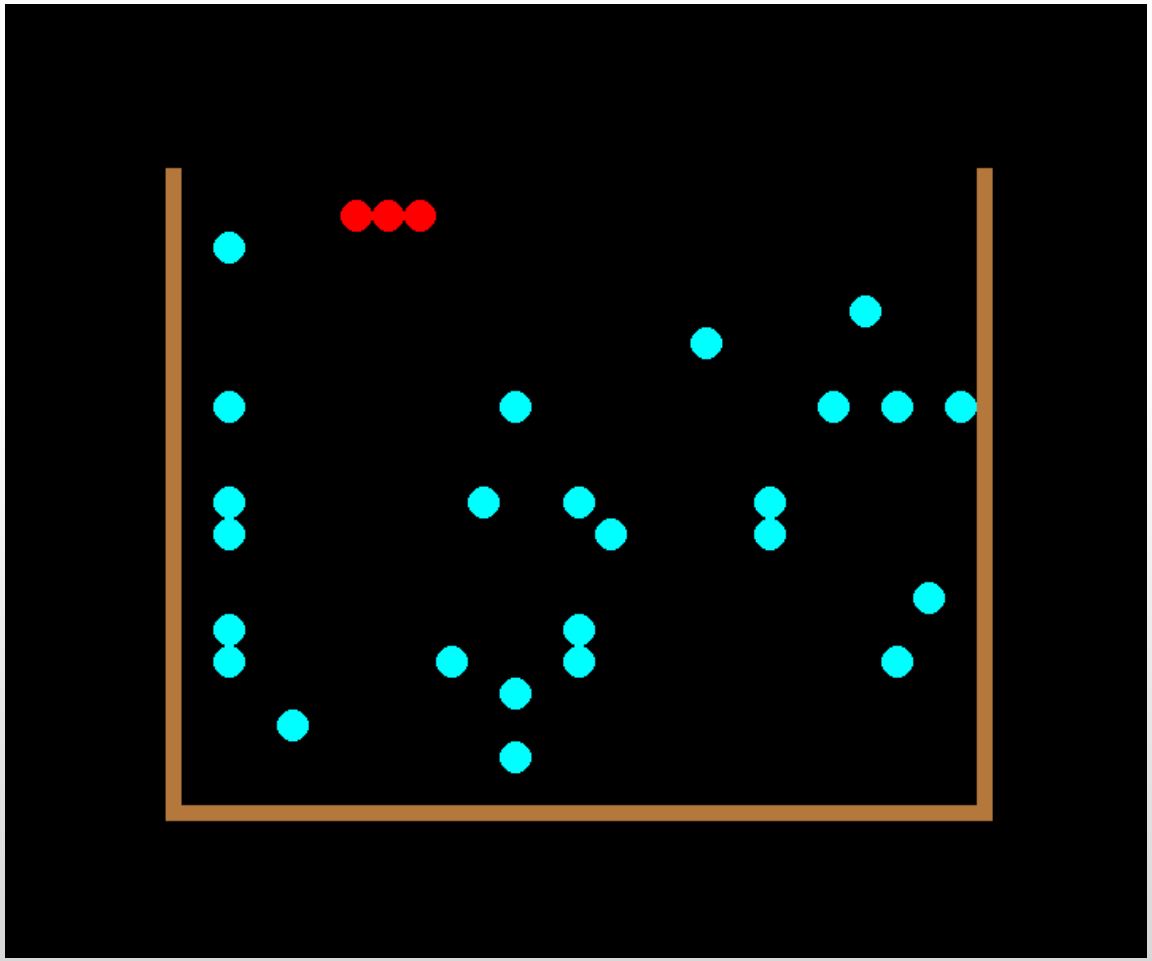
Don’t stand in doorways!



Don’t block the door

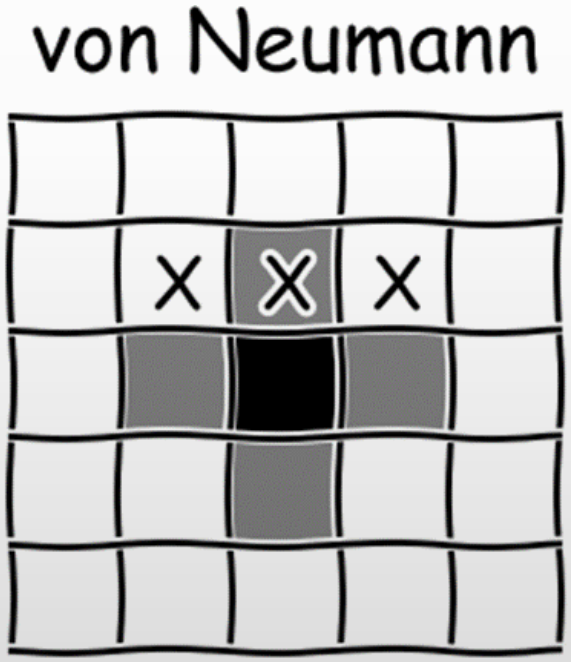
- Don’t forget to do a demo
- Measuring time for everyone out safely
- Using five trials with
  - Blocking took 123.2 updates (pstdev 10.07)
  - Non blocking took 111.0 updates (pstdev 9.47)

Obstacles



One small change

"Just" disallow the grid points with obstacles



Agents with agendas

Obstacles might stop you getting to the door, but what if they are tables with food or drinks?

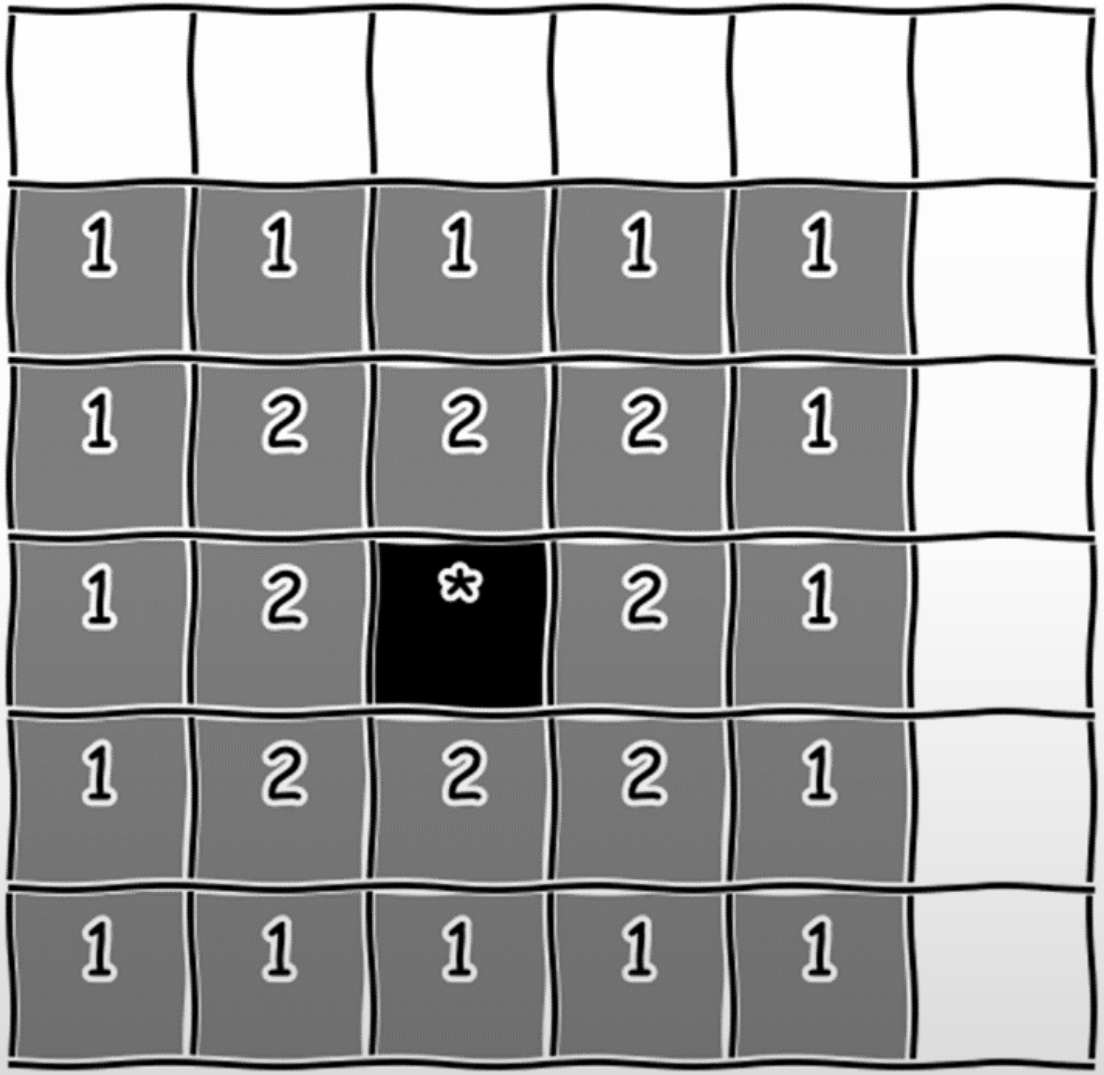
~~One~~ more small change(s)

- Disallow standing on tables
- Give each “table” a temptation value
  - And finite food
- Create a static floor field using temptation

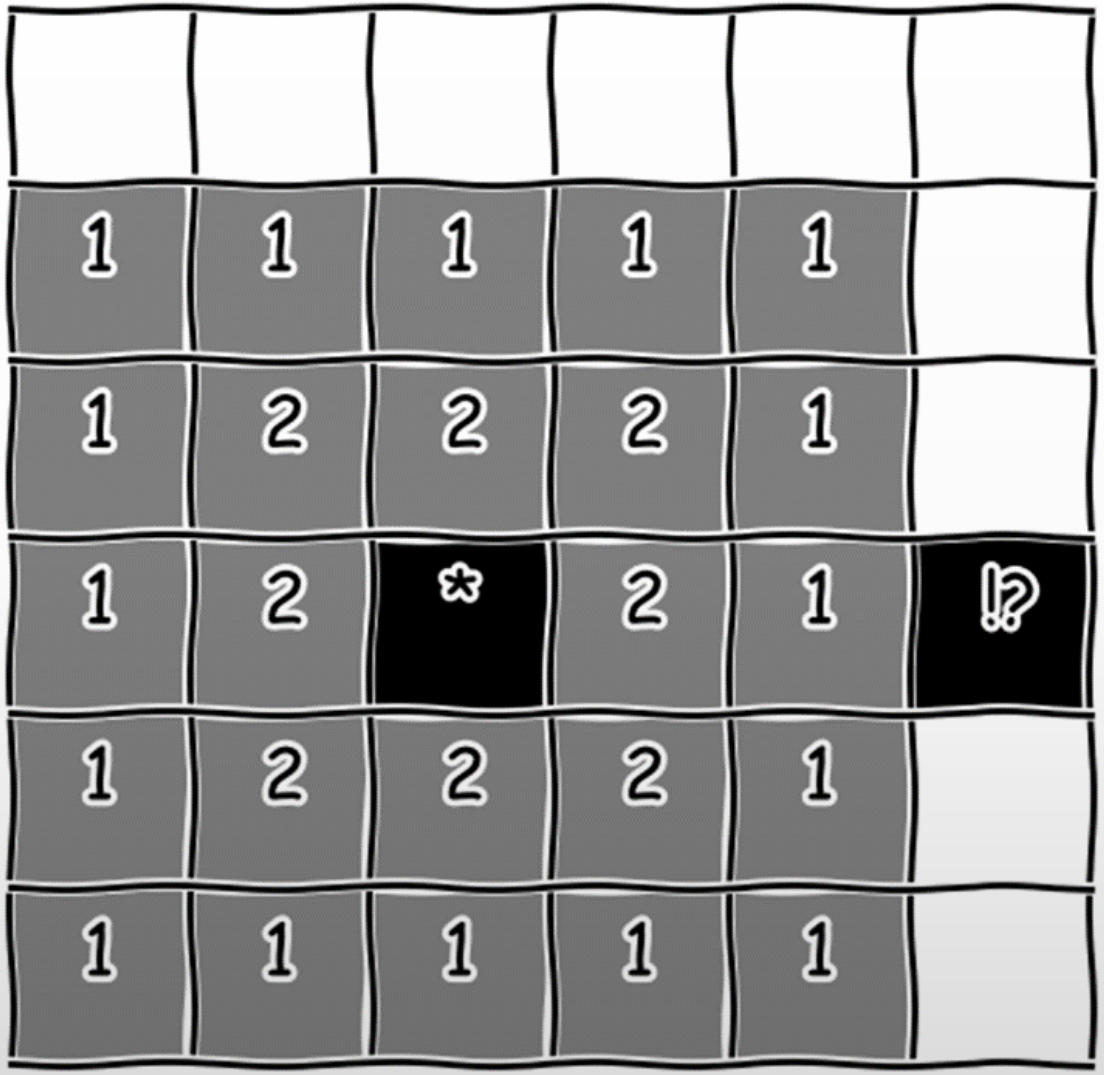
```
std::vector<std::vector<int>> field;
```

- Blob hunger
  - `food_needed`
  - When no longer hungry revert to moving towards the door
- Blob weighting:
  - `field_value * field_value;`

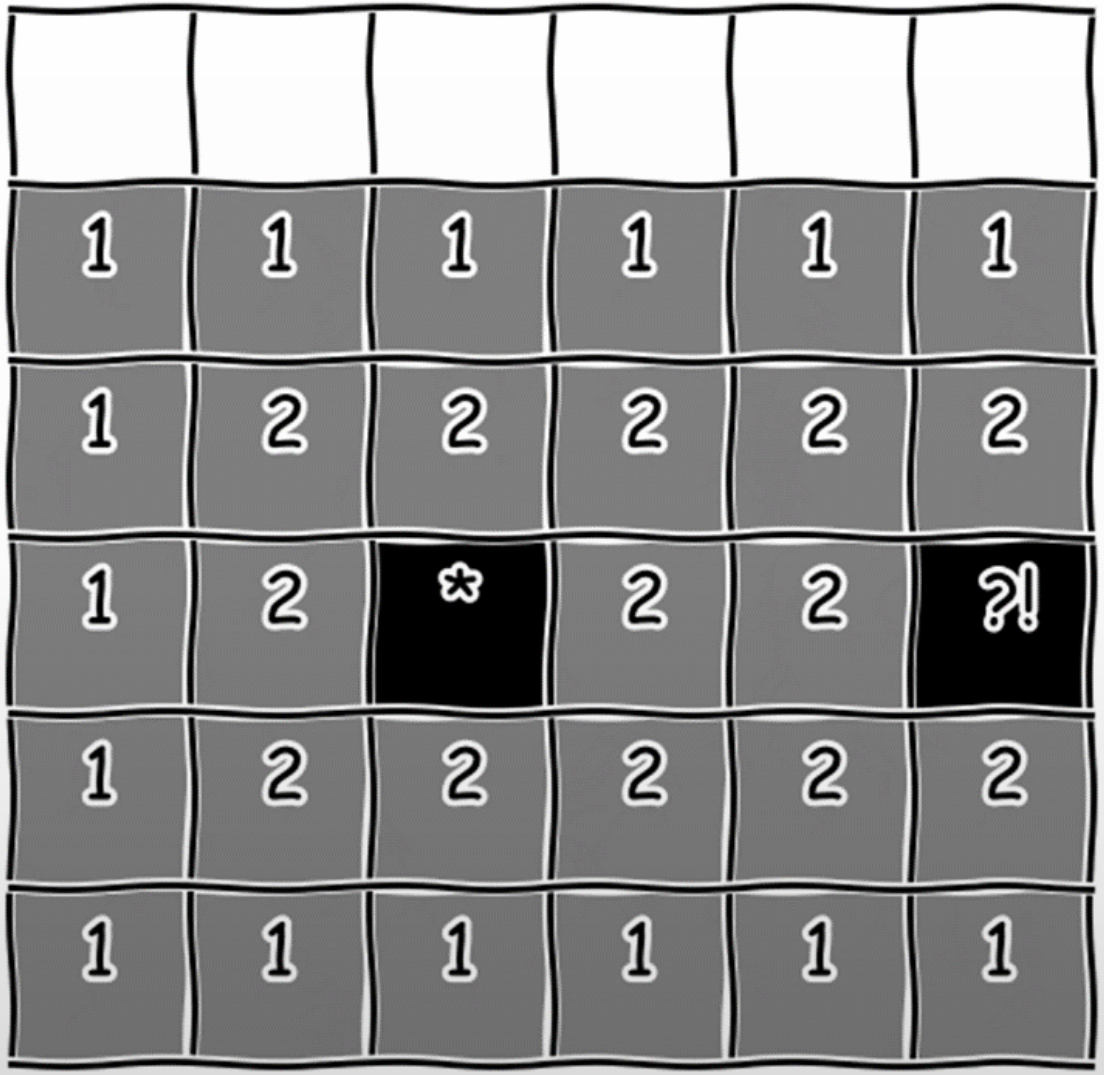
Floor Field



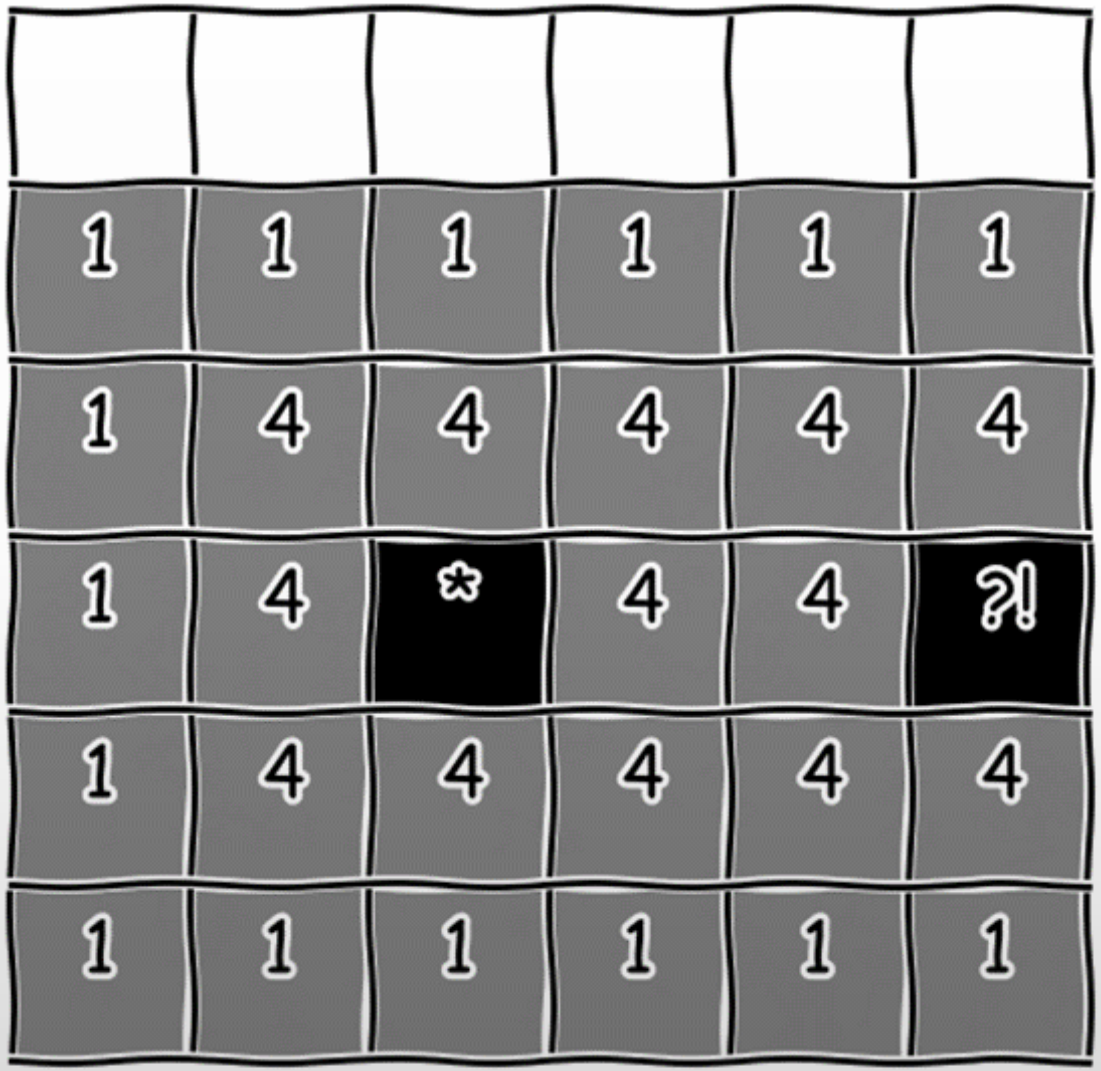
Floor Field



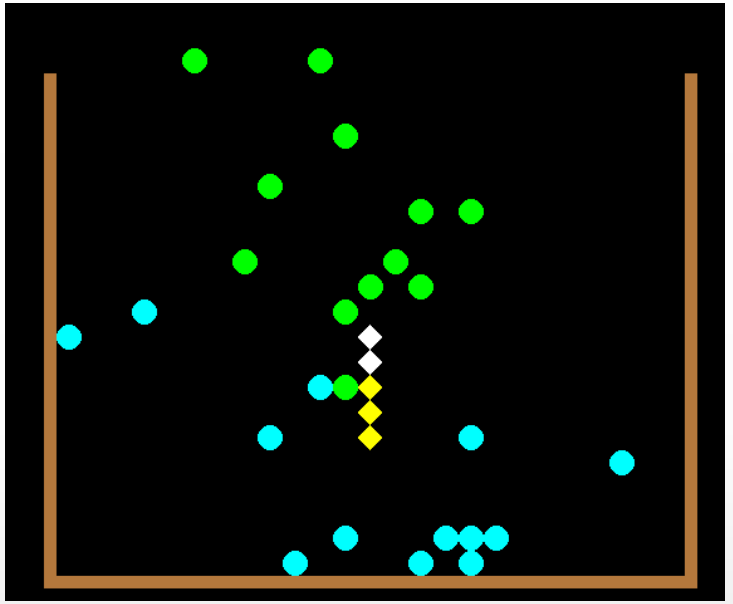
Floor Field



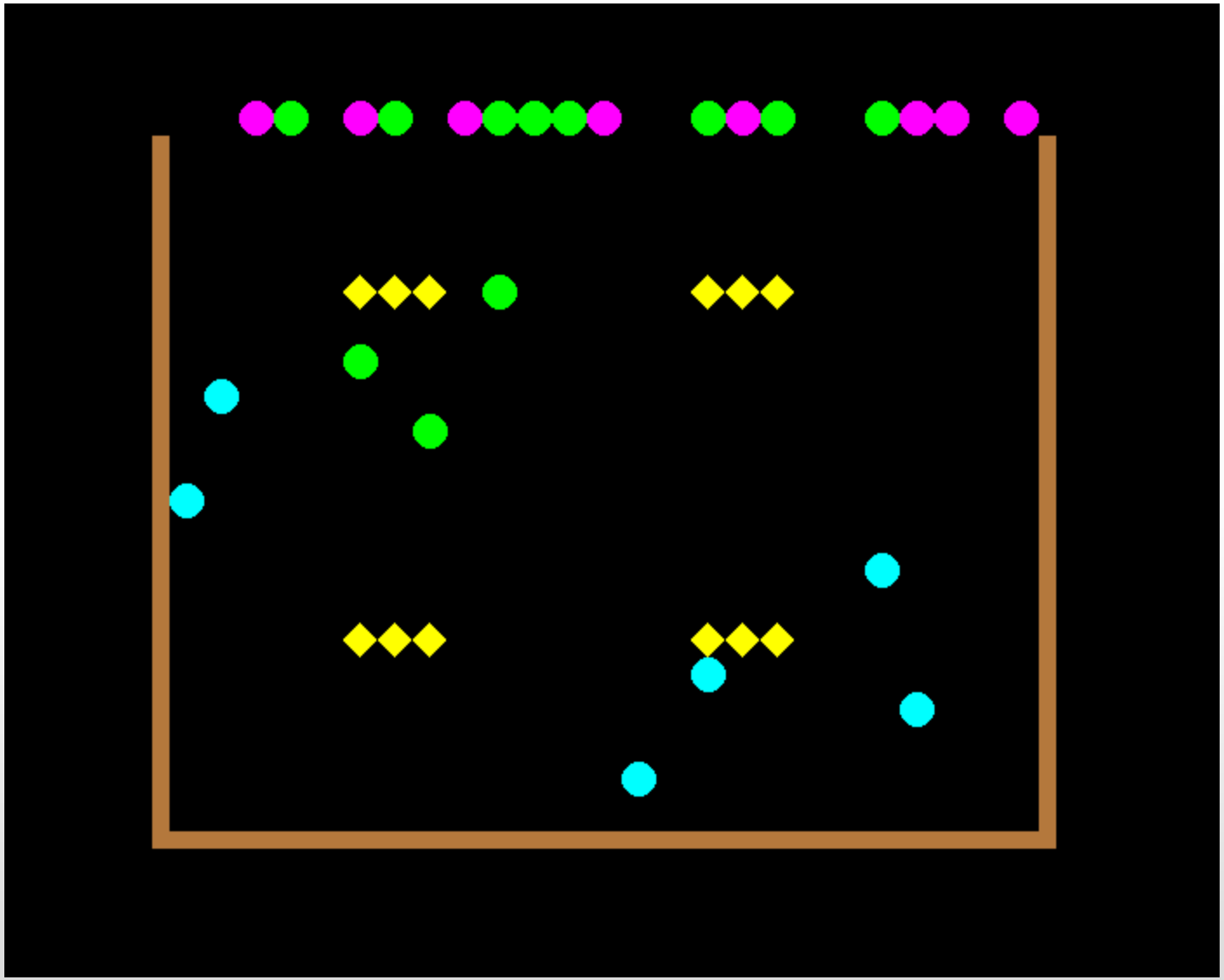
Floor Field ^ 2



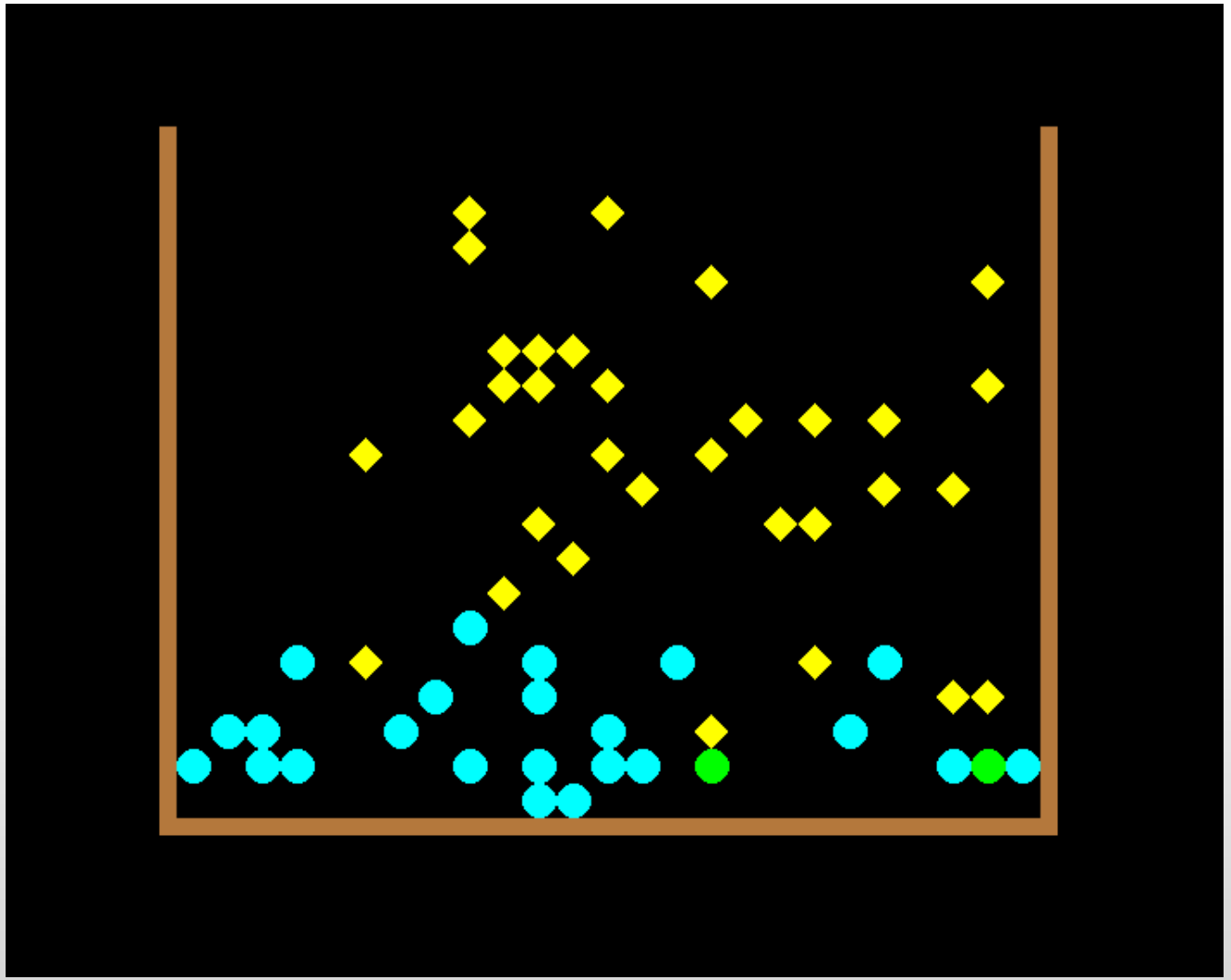
And some options



- block door?
- tables
  - How many? Where? What shape?
- temptation
  - Set up front – think signposts for tables
- food needed
  - Blue blobs hungry, green blobs full (but still keep eating)
- food per table
  - Tables go white when food on them has gone
  - But the floor field is static, so they still attract blobs



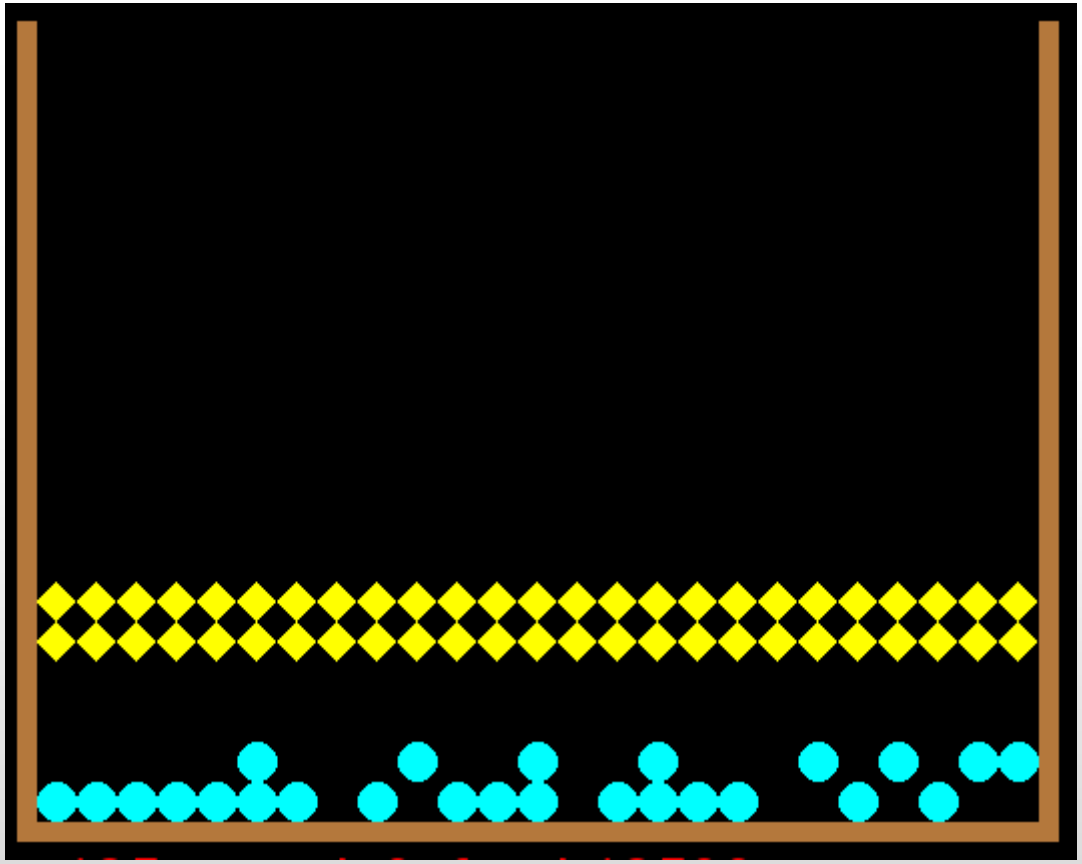
What if we randomly place tables?



Did Fran do a demo?

- Do the empty tables get in the way?
  - And still attracted the hungry blobs
  - Cos the floor field was **static**
- We could remove them and update the floor field
  - **dynamic** floor field
  - Do some demos Fran!

Breakout!



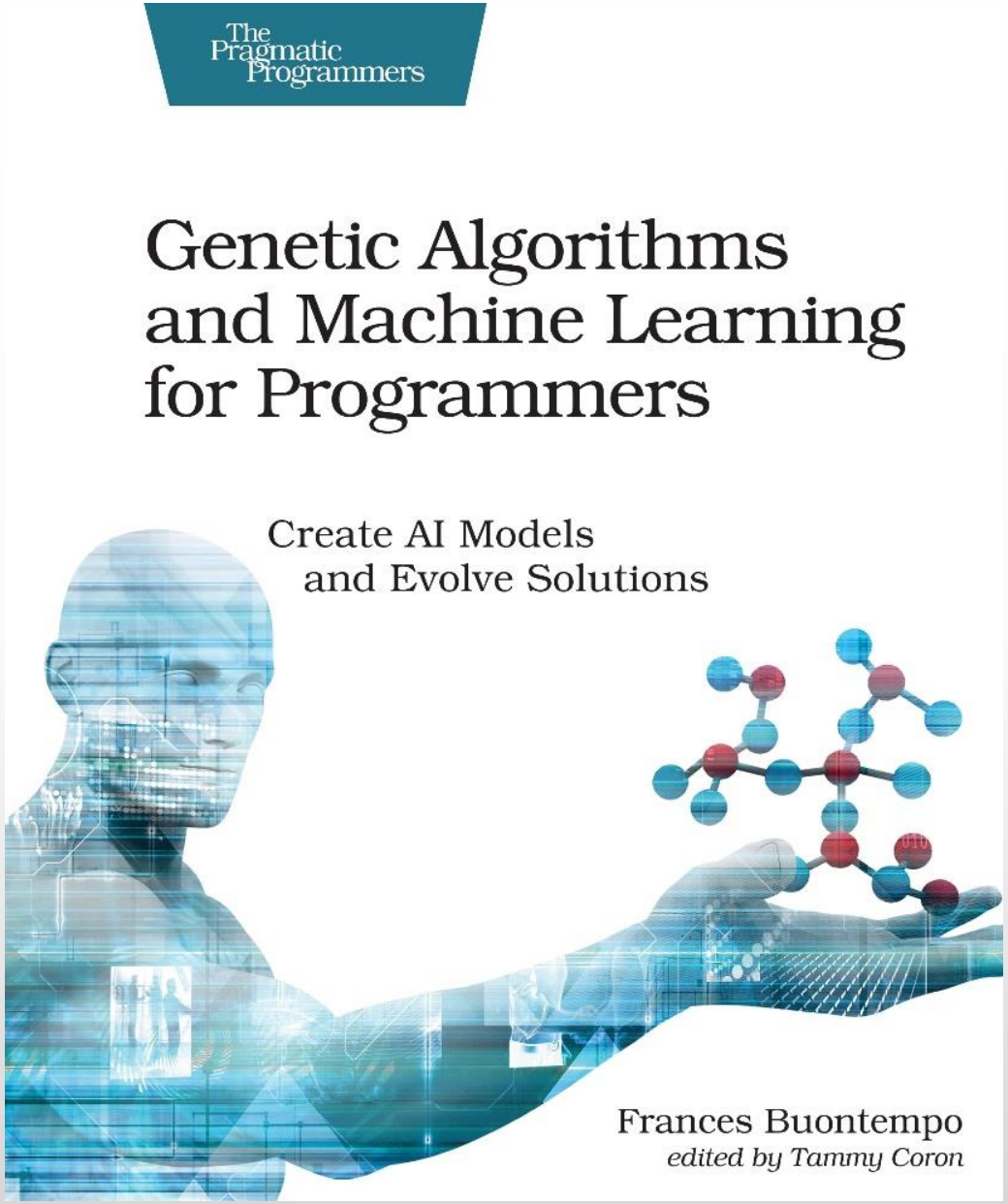
Possible extensions

- Add some who will
  - stop to talk,
  - stand in doorways,
  - try to follow a friend
- Different food or drink preferences/choices
- People re-stocking food
- For now, where shall we put the tables?

Prior Art

- **Genetic algorithms** to design room layout
  - https://www.joelsimon.net/evo_floorplans.html
  - https://itc.scix.net/pdfs/w78-2020-paper-002.pdf
  - We have random table layout
    - And a fitness function – no one is hungry and all get home safe
  - Left as an exercise for the reader/listener
- Donald Knuth used **graph theory** to design an efficient kitchen. He determined what items in the kitchen, toaster, sink, etc. wanted to be close to other items … he found that everything wanted to be close to the wastebasket, so his kitchen is designed with a *centrally located wastebasket* easily accessible from all locations.
  - http://foodnerdkitchen.blogspot.com/2012/06/inspiration-from-donald-knuth.html

What have we learnt?


The Pragmatic Programmers
Genetic Algorithms and Machine Learning for Programmers
Create AI Models and Evolve Solutions
Frances Buontempo
edited by Tammy Coron