

How to diffuse your way out of a paper bag

April 2015

Frances Buontempo

@fbuontempo

frances.buontempo@gmail.com

overload@accu.org

<https://github.com/doctorlove/paperbag>

Copyright Notice

© 2015 Bloomberg L.P. Permission is granted to copy, distribute, and display this material, and to make derivative works and commercial use of it. The information in this material is provided "AS IS", without warranty of any kind. Neither Bloomberg nor any employee guarantees the correctness or completeness of such information. Bloomberg, its employees, and its affiliated entities and persons shall not be liable, directly or indirectly, in any way, for any inaccuracies, errors or omissions in such information. Nothing herein should be interpreted as stating the opinions, policies, recommendations, or positions of Bloomberg.

Objectives

- Think about randomness
- Think about testing
- Draw pictures in C++
- Encourage you to investigate machine learning
- ...or have a fun pet project
- Send articles to Overload@ACCU.org

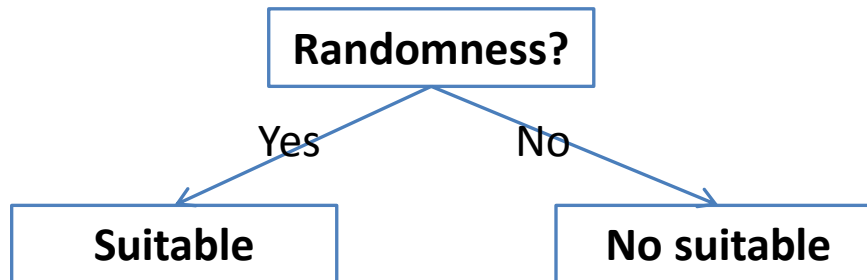
Previously...

- ACCU 2013
- Rule induction

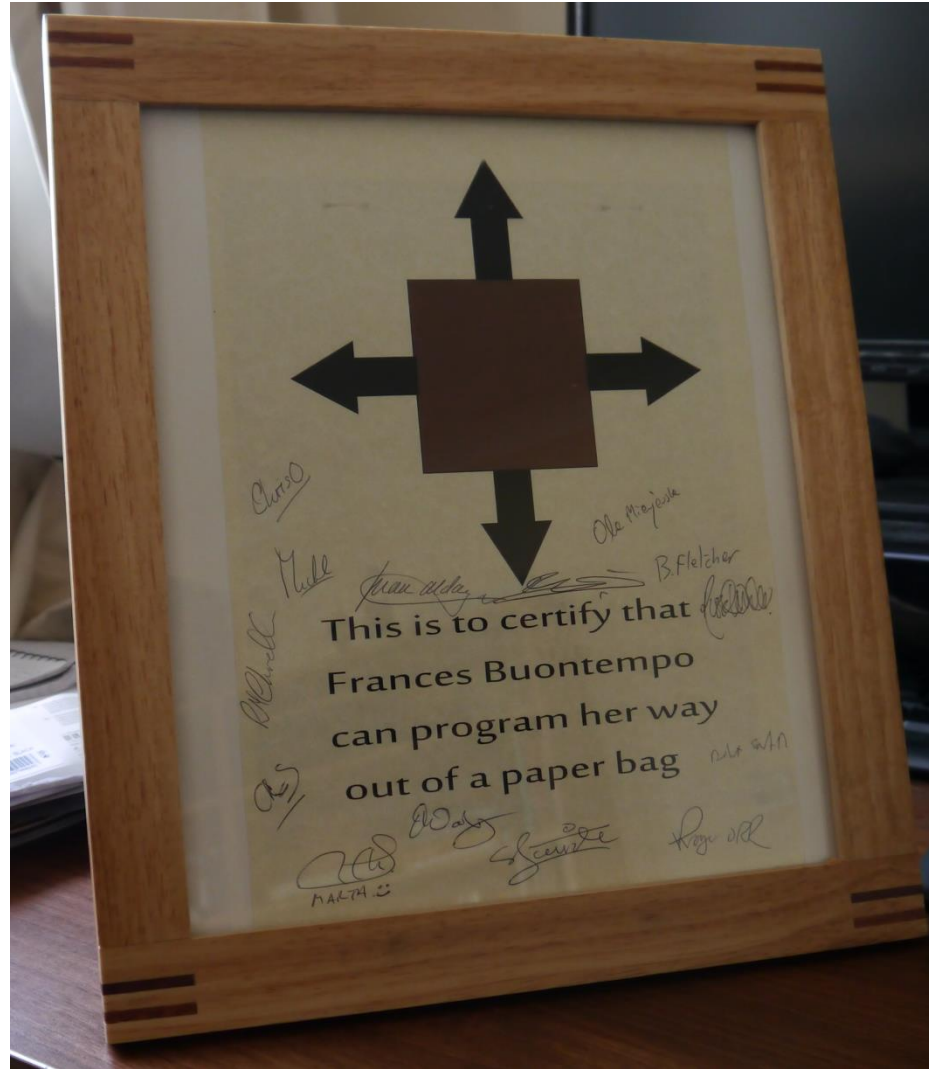
Name	Inputs	Learning	instance-based (lazy) learning	Randomness	Output
regression	numeric	supervised	false	no	Numeric
k-th nearest neighbours	numeric	unsupervised	true	no	data points
kohonen neural network	numeric	unsupervised	false	no	Clusters
feedforward nn	numeric	supervised	false	no	Numeric
recurrent nn (eg Hopfield)	binary	reinforcement	true	no	State
C4.5 or See5	categoric	supervised	false	no	decision tree
CART	any	supervised	false	no	Tree
genetic algorithm	any	unsupervised	false	yes	Solution
dendral	numeric	hypothesis formation	false	no	expert system (possible chemical structures)
ACO	spatial	unsupervised	false	yes	best 'path'

We learnt

- Once we added a “Suitable?” column
- C4.5 from <https://www.rulequest.com/>



Award!



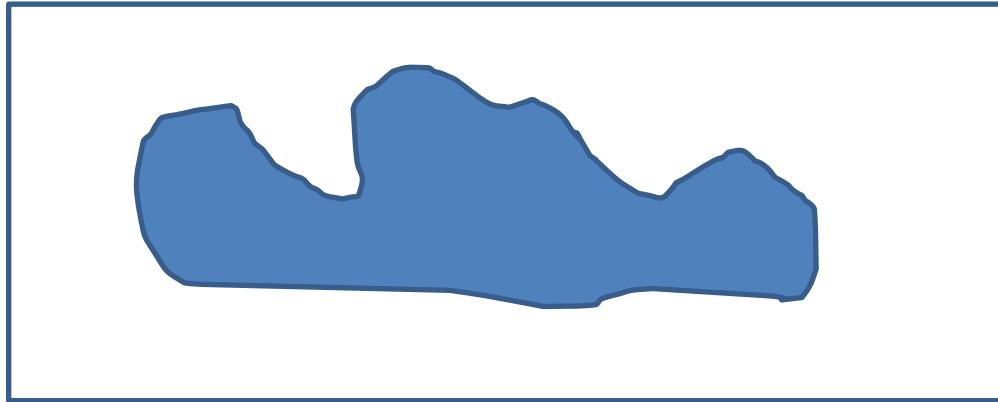
Previously...

- Over-engineered – using machine learning
 - Ant Colony Optimisation
 - http://accu.org/content/conf2013/Frances_Buontempo_paperbag.pdf
 - Genetic algorithms
 - <http://accu.org/index.php/journals/1825>
 - Particle swarm optimisation
 - <http://accu.org/index.php/journals/2023>

Now...

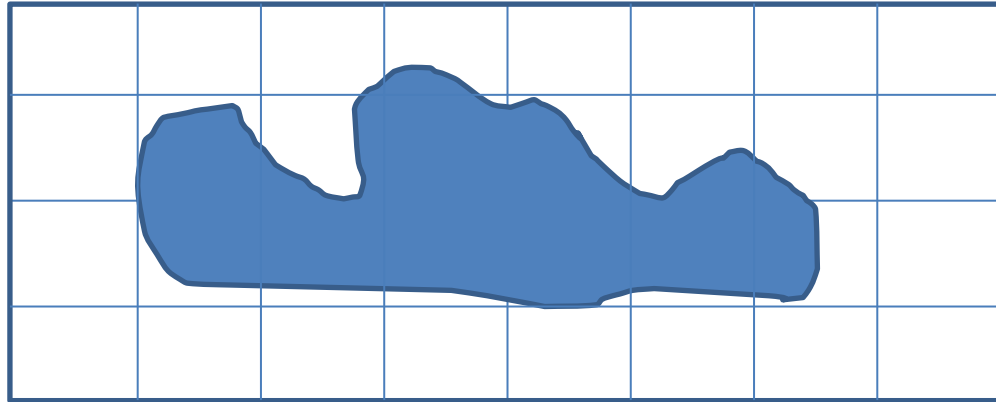
- Approaches used randomness
 - As indicated by C4.5
- Some need a model
 - e.g. genetic algorithms
- Time for “Monte Carlo” simulation(s)
 - Generate a few possible results and see what happens
 - Usually we average the results to get an answer

What's MC good for?



What's the area in the squiggle?

Intractable, but guesstimateable



...32 squares:

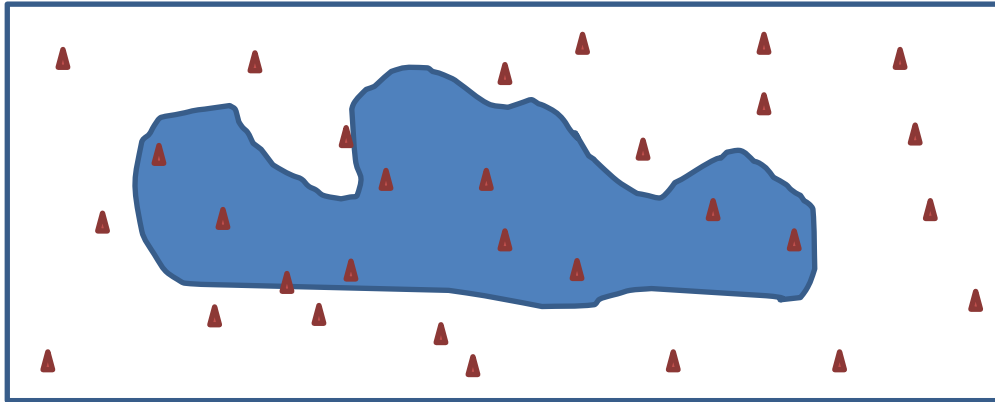
18 (more than $16/32$) with none =>

< 50% of rectangle

Some not empty =>

> 0%

Sampling



“Throw darts” – 10 out of 30 in blob gives approx. 30% of area

What ... ?

- Simulate and see
 - Solve a problem approximately... “optimisation”
 - What happens if... “simulation”
- Spread of infectious disease simulation
 - <http://plus.maths.org/content/mathematics-diseases>
- Finance modelling
- Finding out π (Buffon’s needle etc)

Monte Carlo simulation

Invented in the late 1940s by [Stanislaw Ulam](#), while he was working on nuclear weapons projects at the [Los Alamos National Laboratory](#).

http://en.wikipedia.org/wiki/Monte_Carlo_method

Card sharks...

- Casinos, cards.... MONTE CARLO
- “The first thoughts and attempts I made to practice [the Monte Carlo Method] were suggested by a question which occurred to me in 1946 as **I was convalescing from an illness and playing solitaires**. The question was what are the chances that a [Canfield solitaire](#) laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and **simply observe and count** the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain **differential equations into an equivalent form interpretable as a succession of random operations**. Later [in 1946], I described the idea to [John von Neumann](#), and we began to plan actual calculations. “

http://en.wikipedia.org/wiki/Monte_Carlo_method

What's this got to do with diffusion?



Latin *diffusionem* 'a pouring forth',
stem of *diffundere* 'scatter, pour out',
from *dis-* "apart, in every direction"
+ *fundere* "pour".

<http://www.etymonline.com/index.php?term=diffusion>

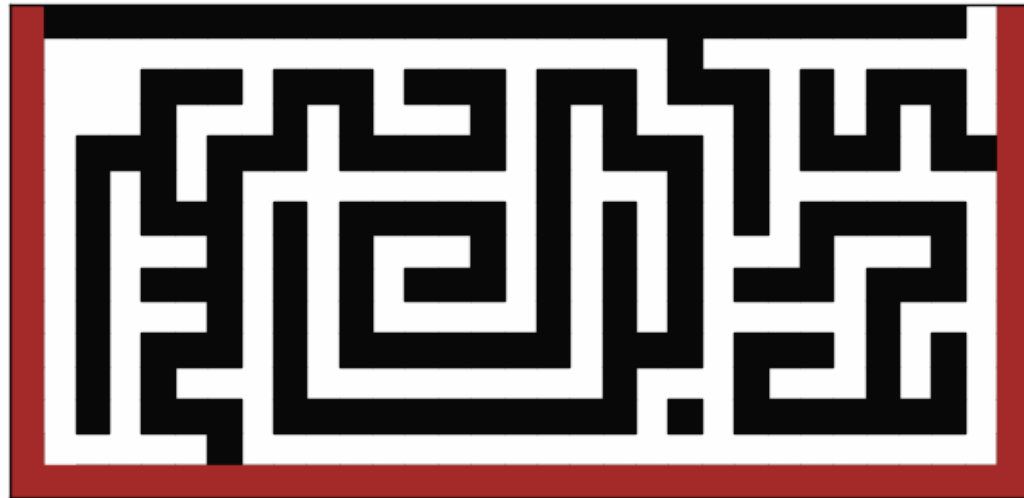
Diffusion

- In gas, water, solid
- Brownian motion
- Geometric Brownian motion
- Jump diffusion
- All about things **spreading out**
 - <http://lben.epfl.ch/files/content/sites/lben/files/users/179705/Simulating%20Brownian%20Motion.pdf>

What's this got to do with paper bags?

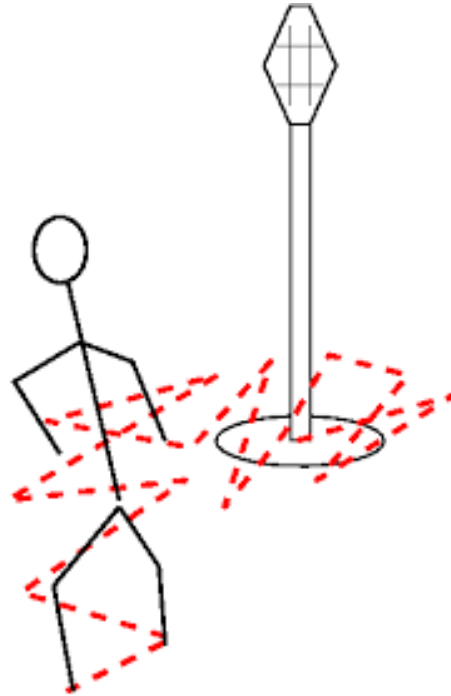
- Use a model (differential equation)
 - Brownian motion
- Questions
 - Can we go off the screen?
 - Can we bust out the bottom or side of the bag?
- What bag?
 - Level of indirection...
 - What colour is it anyway?

Level of indirection



Our model

Random walk



<http://www.physics.ucla.edu/~chester/TECH/RandomWalk/>
http://en.wikipedia.org/wiki/Random_walk

Brownian motion in words

Brownian motion is a stochastic process whose increments are independent, stationary and normal, and whose sample paths are continuous

- Independent – non-overlapping time periods are independent
- Stationary – don't vary with time
- Normal – the increments are normally distributed
- Continuous – can draw without taking your pen off the paper

Brownian motion again

- Markov – no memory
 - next state depends only on current state
- Martingale – $\mathbb{E}[S(t)] = 0$
- Quadratic variation - $\mathbb{E}[S(t)^2] = t$
 - The increment scales with the square root of the time step, otherwise it goes to infinity in finite time
 - Depends on whether the bag is porous or not 😊

Brownian motion in English

Imagine

- Walking, one step at a time
 - For various step sizes
 - Normal (or Gaussian) to be Brownian motion
- Every which way
 - All equally likely
- In various directions
 - AKA a “drunkards walk” = no memory! (Markov)

Model

- Describe the position of a particle in a 2D bag as (x, y)
- $\Delta x = \sqrt{t}dW, \Delta y = \sqrt{t}dW$
 - \sqrt{t} square root of time step, t
 - dW – Normal “random” increment $N(0, 1)$

$$x += \Delta x$$

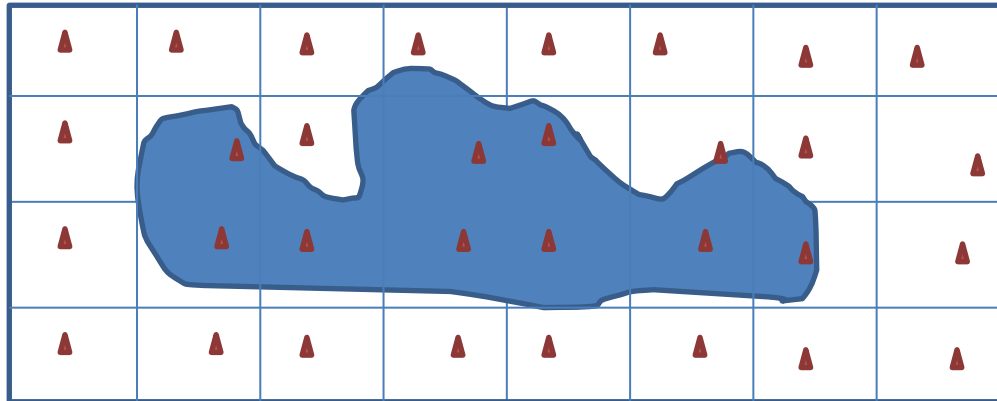
$$y += \Delta y$$

Random increment

- Stochastic *means* random(ish)
- rand
- Uniform random
- “Normal” random
- What is a Mersenne twister anyway?

Why random?

- Low discrepancy numbers instead?



God does not play dice*

- Though particles appear to move randomly, they do not.
- Not even simulated ones – which use “deterministic” random number generators



*Except in the movies and our current physics models

How do I get a random number?

```
int GetRandom()
{
    return 59;
    /*A perfectly randomly picked number*/
}
```

<http://stackoverflow.com/questions/4195958/how-do-i-scale-down-numbers-from-rand>

How random is the random play on your music player? If it played the same song a few times running you'd be annoyed.

<http://www.bbc.co.uk/news/technology-31302312>

Random...

- rand() not recommended for serious random-number generation
- Implementation-defined whether rand() is thread-safe
- rand considered harmful -
<http://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful>
- “Some pseudo random number generators lead to inconsistent results in random walk simulations”
 - Ferrenberg affair: see
http://www.academia.edu/4222855/Pseudo_Random_Coins_Show_More_Heads_Than_Tails

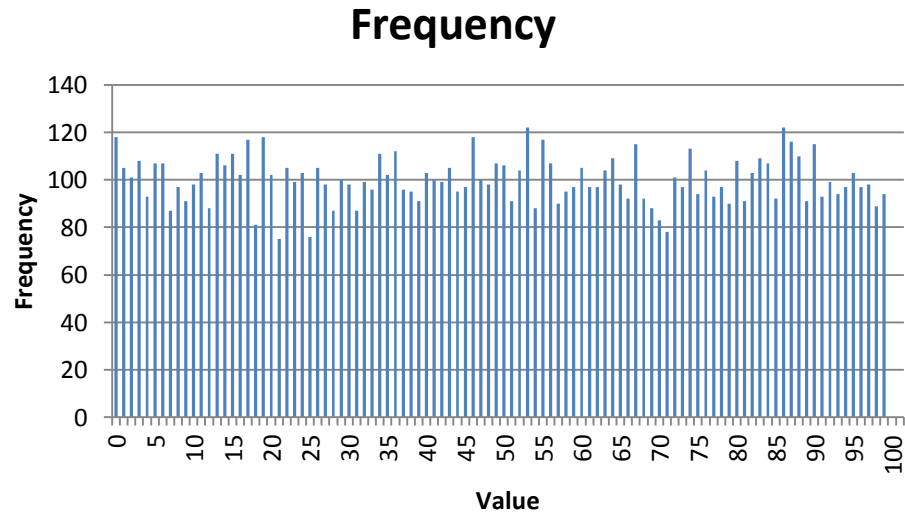
How do I get a number in [0, 100]?

```
int value = rand() % 100;
```

```
int value = (rand() * 1.0/RAND_MAX) * 100;
```

```
int value;  
do {  
    value = rand();  
} while (value > 100);
```

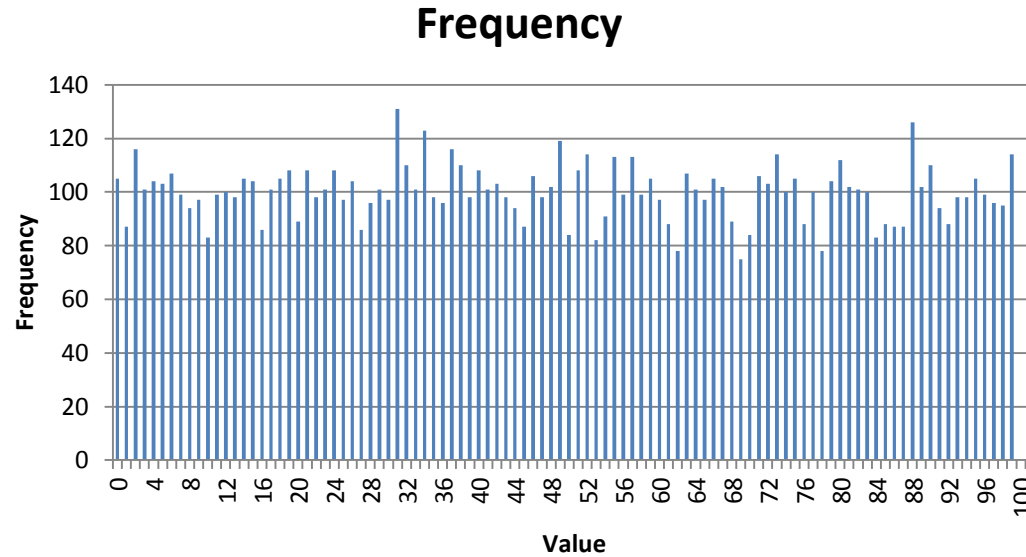
rand() % 100



Oops...

Value	Freq
100	0

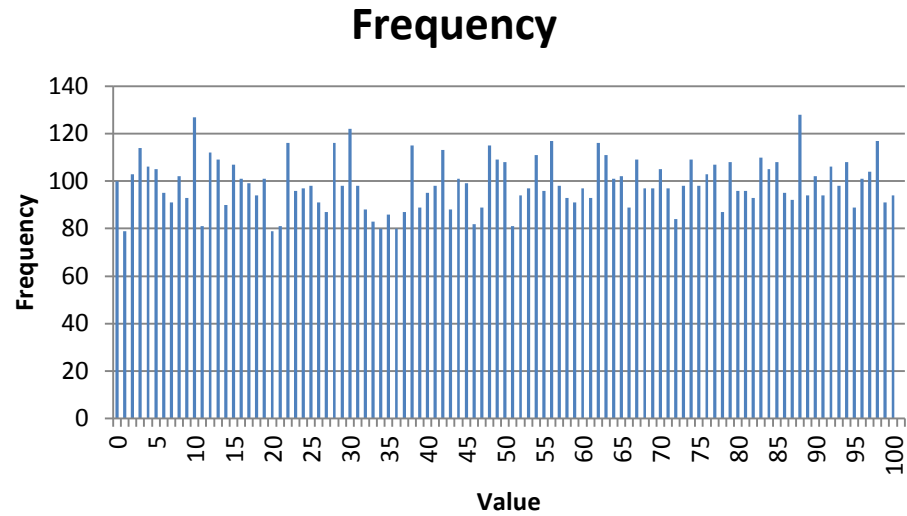
$$(\text{rand}() * 1.0 / \text{RAND_MAX}) * 100$$



Oops...I did it again

Value	Freq
100	0

Rejection sampling

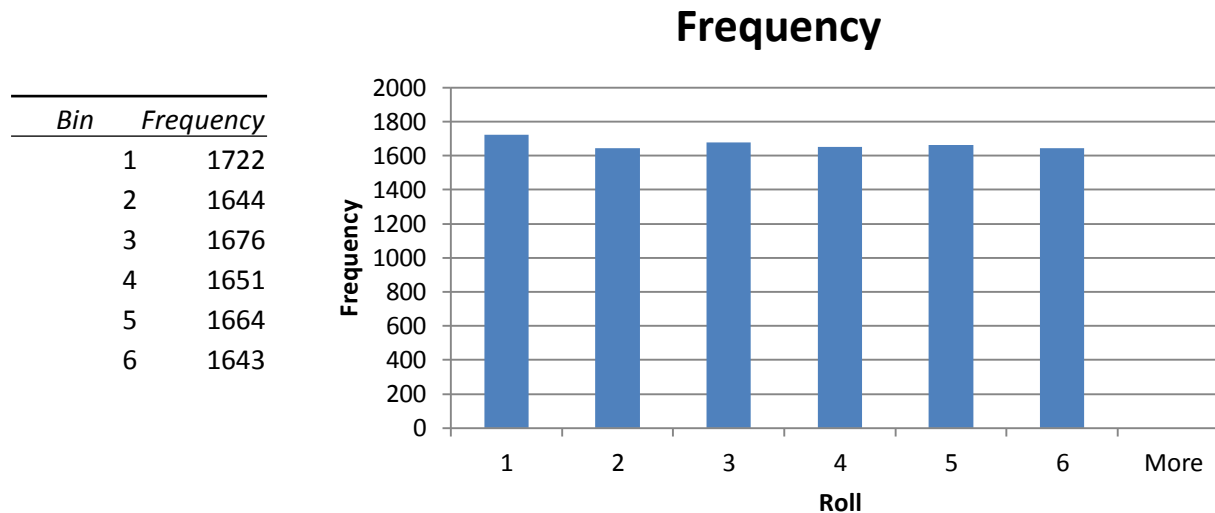


Value	Freq
100	94

So what?

Dice rolling... [1, 6] doing [0, 6) then add 1:

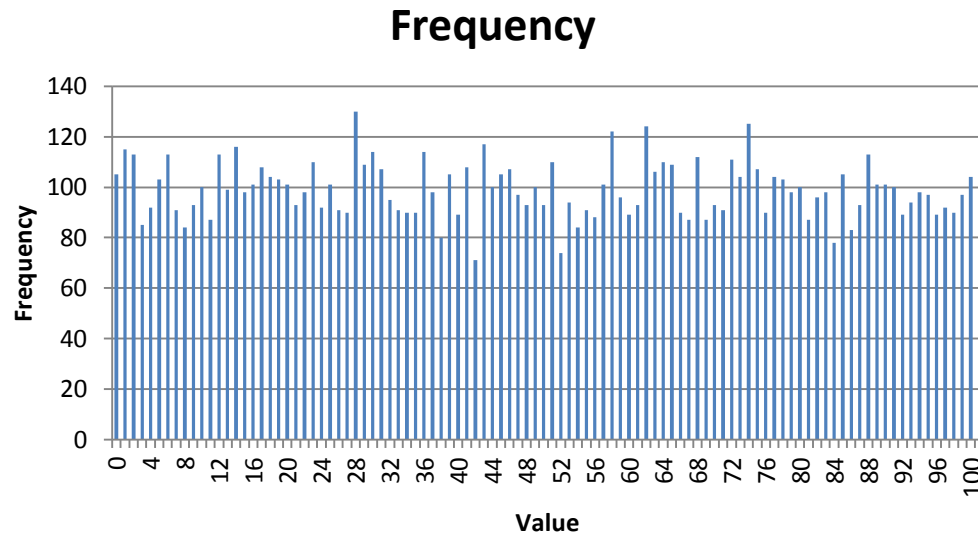
```
((rand() * 1.0/RAND_MAX) * 6) + 1;
```



Bonus Question – can this ever be 7?

New, improved

```
#include <random>
std::random_device rseed;
std::mt19937 rgen(rseed()); // mersenne_twister
std::uniform_int_distribution<int> dist(0,100);
int value = dist(rgen);
```



Value	Freq
100	104

Mersenne twister

- Better than rand
- Mersenne prime – a prime of the form
$$M_n \equiv 2^n - 1$$
- Uses matrix linear recurrence over a finite binary field. Fast, “high quality” pseudorandom numbers
- See M. Matsumoto and T. Nishimura,
Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator,
ACM Transactions on Modeling and Computer Simulation,
Vol. 8, No. 1, January 1998, pp. 3-30
<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>.

Linear congruential

$$x_{n+1} = (ax_n + c) \bmod m$$

- Linear $y = mx + c$
- Congruential $\bmod m$
- See Knuth “The Art of Computer Programming Vol 2”

But why?

- Originally called "Primitive Twisted Generalized Feedback Shift Register Sequence"
- Knuth said "The name is a mouthful"
- Reply: 'How about "Mersenne Twister?" Since it uses Mersenne primes, and it shows that it has its ancestor Twisted GFSR. '
- **'It sounds like a jet coaster'**

So what?

- Recall: we want Normal random increments
 - over finite steps the increments are normally distributed
- How?
 - $(\sum_{i=1}^{12} \psi_i) - 6$ where ψ_i are independent uniform random variable in $[0, 1]$
 - Page 587 “Paul Wilmott Introduces Quantitative Finance”
 - Box-Muller $\sqrt{-2 \ln x_1} \cos(2\pi x_2)$ for x_1, x_2 uniform $[0,1]$ with x_1, x_2 on the unit circle
 - Chapter 7, “Numerical Recipes in C” Press et al

C++11, simples

```
std::random_device rseed;  
std::mt19937 rgen(rseed());  
std::normal_distribution<double> dist;  
int value = dist(rgen);
```

std::random_device is a uniformly-distributed integer random number generator that produces non-deterministic random numbers.

std::random_device may be implemented in terms of an implementation-defined pseudo-random number engine if a non-deterministic source (e.g. a hardware device) is not available to the implementation. In this case each std::random_device object may generate the same number sequence.

http://en.cppreference.com/w/cpp/numeric/random/random_device

Finally, some code

```
template<typename Shape, typename T = double>
class Particle {
public:
    Particle(Shape shape, T root_t, std::function<void(Shape &)> draw,
            T x = 0, T y = 0, unsigned int seed = 1);

    void Update() {
        x += move_step(); y += move_step();
        //possibly check the edges
        shape.setPosition(x, y);
    }

    void Draw() { draw(shape); }

private:
    Shape shape; T x; T y; T root_t;
    std::function<void(Shape &)> draw;
    std::mt19937 engine;
    std::normal_distribution<T> normal_dist;

    T move_step() { return root_t * normal_dist(engine); }
};
```


Now for some pictures...

```
#include <SFML/Graphics.hpp>    // http://www.sfml-dev.org/

int main() {
    sf::RenderWindow window(sf::VideoMode(800, 600), "My window");

    while (window.isOpen()) {
        // check all the window's events
        sf::Event event;
        while (window.pollEvent(event)) {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear(sf::Color::Black);

        // draw everything here...
        // window.draw(...);


        window.display();
    }
}
```

Drawing in C++

```
bool paused = false;
while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window.close();
        if (event.type == sf::Event::KeyPressed)
            paused = !paused;
    }

    window.clear();
    drawBag(window, left, right, base);
    for(auto & particle: particles) {
        if (!paused)
            particle.Update();
        particle.Draw();
    }
    window.display();

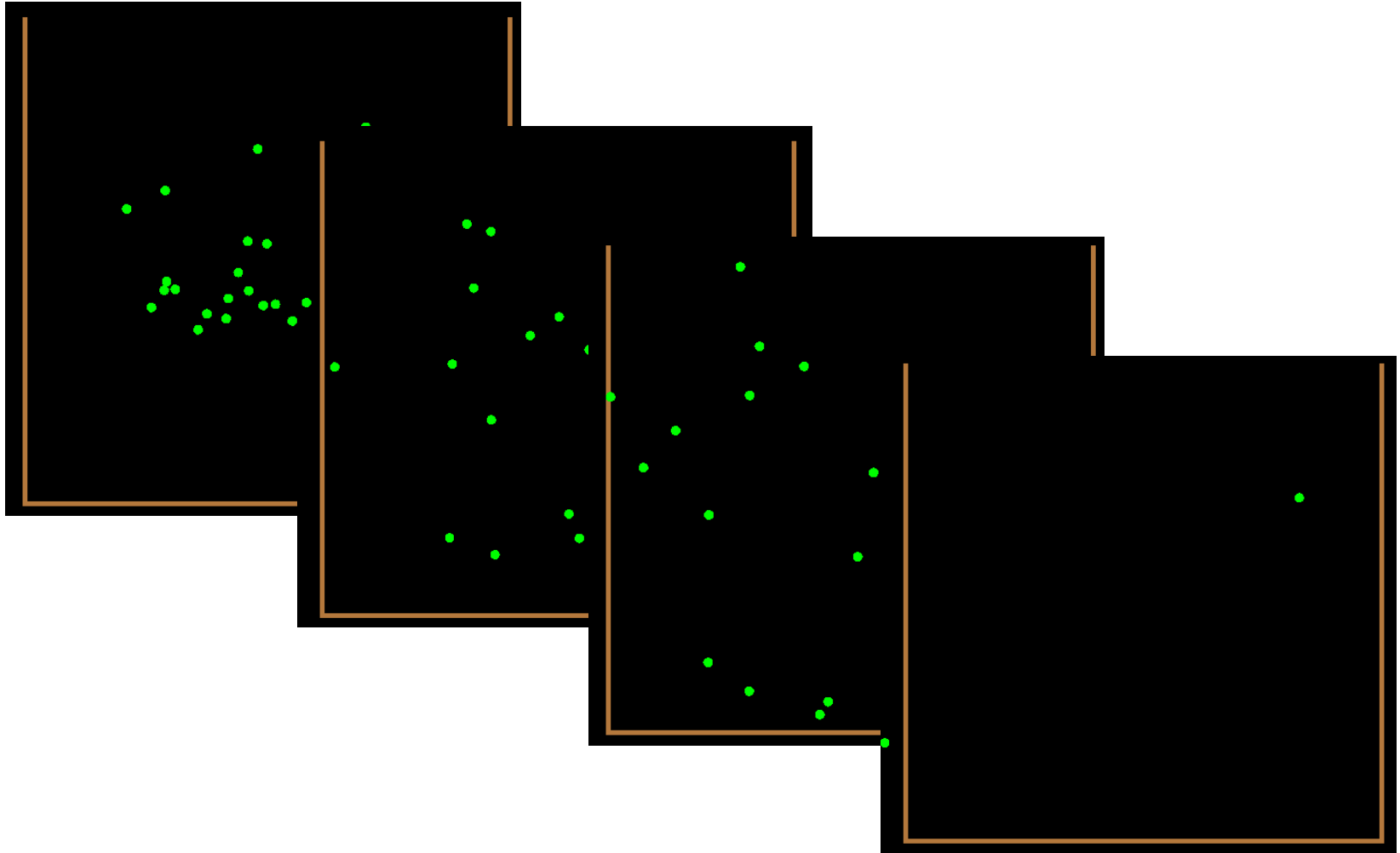
    std::this_thread::sleep_for(std::chrono::milliseconds(50));
}
```



Action...

- (Note to self – do a demo)

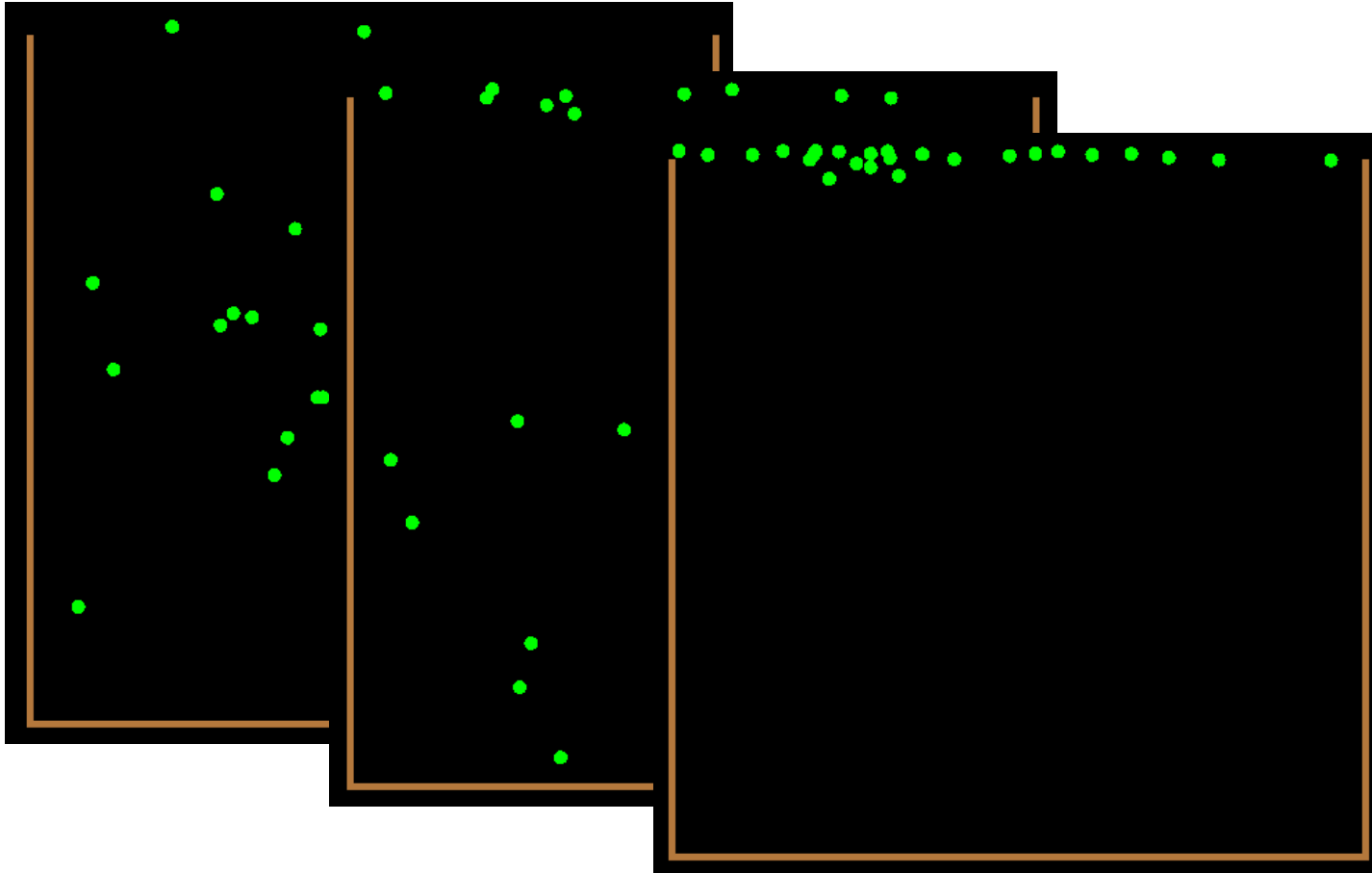
Eventually...



Question

- Can the particles go through the side of the bag?
- Should they be allowed back in again?

Non-permeable



Assumptions

- The paper bag is “permeable” – at least not full of concrete
- This is conducted above absolute zero
- I assume some particles escaping is good enough – they do all eventually go... eventually

Stop hacking and think

- It takes a..g..e..s..
- Can I just set fire to the bag? Not today.
- We want to mostly go up.
- Stock prices (almost) always go up, right?
- Let's try something different
 - A stock price simulation

But where's the paper bag?

- Level of indirection
- Put the stock price simulation in a bag
- New model: Geometric Brownian motion
 - A little more complicated than Brownian motion

Failing that



Stock prices

- Supply/demand
- “Not a prediction, but a convention” Keynes
- So let’s predict them using Monte Carlo simulations
 - Need a model - GBM
 - Based on the “efficient market hypothesis” i.e. it’s impossible to “beat the market” or make money from it.

<http://www.investopedia.com/articles/07/montecarlo.asp>

GBM

Geometric Brownian motion is a

“continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion (also called a Wiener process) with drift.” http://en.wikipedia.org/wiki/Geometric_Brownian_motion

$$S_t = S_0 e^{\mu t + \sigma f(t)}$$

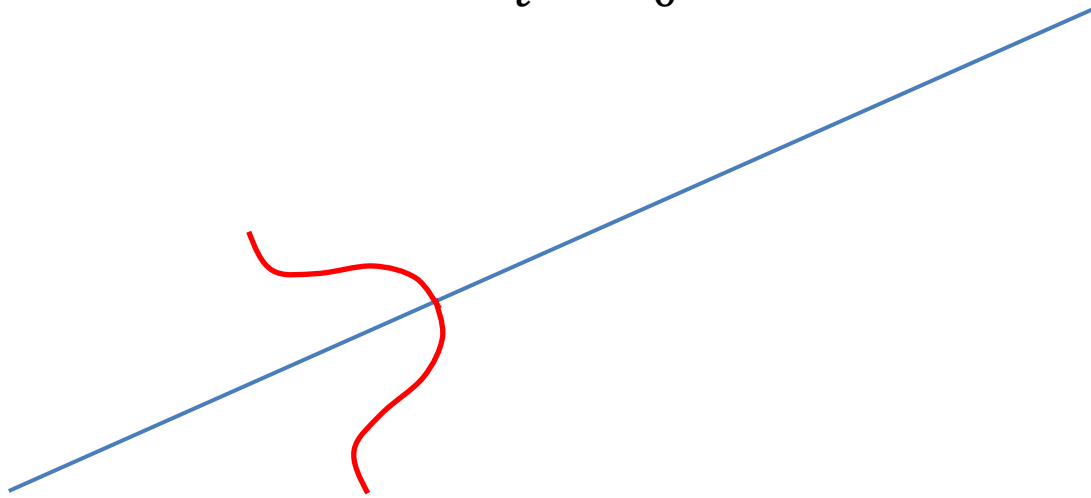
Model

- $dS = S(\mu dt + \sigma dz)$ in continuous time
 - An Itô process
- $\Delta S = S(\mu \Delta t + \sigma \sqrt{\Delta t} dW)$
 - μ , drift
 - Δt , time step (discrete approximation) to dt
 - σ , volatility
 - dW – “random” increment $N(0,1)$
- Called a generalised “Wiener” process -
http://en.wikipedia.org/wiki/Wiener_process

Drift and shock

$$S += S(\mu dt + \sigma \sqrt{dt} dW)$$

n.b. closed form $S_t = S_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma W_t}$



What values to use?

- Drift
 - risk free rate?
 - LIBOR rate?
- Volatility
 - historical?
 - backed out (implied) from current prices?
 - GARCH models?
- Make it up 😊

Finally, some code

```
class PriceSimulation {
public:

    PriceSimulation(double price, double drift, double vol,
                    double dt, unsigned int seed);

    double update();

private:
    std::mt19937 engine;
    std::normal_distribution<> normal_dist;
    double price;
    double drift;
    double vol;
    double dt;
};
```


Update

```
double PriceSimulation::update()  
{  
    double stochastic = normal_dist(engine);  
  
    double increment =  
        drift * dt + vol * sqrt(dt) * stochastic;  
  
    price += price * increment;  
    return price;  
}
```

Things can go wrong

- What seed to use?

- VS2012 won't accept 0

- <https://connect.microsoft.com/VisualStudio/feedback/details/776456>

- Should each price simulation have its own generator?

- Using

```
unsigned int seed =  
std::chrono::high_resolution_clock::  
    now().time_since_epoch().count();
```

in a tight loop is a Bad Idea

- `std::random_device` can be a bad idea

- <http://sourceforge.net/p/mingw-w64/bugs/338/>

- <http://stackoverflow.com/questions/18880654/why-do-i-get-same-sequence-for-everyrun-with-stdrandom-device-with-mingw-gcc4>

How do we test this?

```
#define CATCH_CONFIG_MAIN
#include <catch.hpp> //https://github.com/philsquared/Catch.git
#include <random>
#include "../Lib/PriceSimulation.h"

TEST_CASE("Test that random numbers have mean 0", "[Simples]")
{
    std::mt19937 engine;
    std::normal_distribution<> normal_dist;

    REQUIRE(normal_dist.mean() == 0.0);
}

TEST_CASE("Test that random numbers have standard deviation 1", "[Simples]")
{
    std::mt19937 engine;
    std::normal_distribution<> normal_dist;

    REQUIRE(normal_dist.stddev() == 1.0);
}
```

Or...

```
TEST_CASE("Test that stock price simulation starting  
at 0 remains at 0", "[Property]")  
{  
    const double start_price = 0.0;  
    const double dt          = 0.1; //or whatever  
    const unsigned int seed  = 1;  //or whatever  
    MC101::PriceSimulation price(  
        start_price, 0.3, 0.2, dt, seed);  
  
    REQUIRE(price.update() == 0.0);  
}
```

Recurse

- Should I randomly seed the seed?
- Randomly generate the numbers that don't matter?
- Can I use Monte-Carlo simulations to test my simulation code?
- YES!!!!!!

Go on then

- Property based testing
 - AKA “axiom based” or “specification based” testing
- QuickCheck (Haskell), ScalaCheck, FsCheck
- Can get abstract quickly – e.g. concepts and axioms:

http://www.jot.fm/issues/issue_2011_01/article10.pdf

Test properties

- Test that stock price simulation starting at 0 remains at 0
- Test that the returns have the requested drift
- Etc etc
- Avoids “magic” numbers
- Might find edge cases you hadn’t thought of
 - Potential unit tests

QuickCheck

```
#include "quickcheck/quickcheck.hh" //http://software.legiasoft.com/git/quickcheck.git
#include <vector>
```

```
using namespace quickcheck;
```

```
typedef std::vector<int> Vector;
```

```
class PReverseCancelsReverse : public Property<Vector> {
    bool holdsFor(const Vector& xs) {
        Vector ys = xs;
        reverse(ys.begin(), ys.end());

        return xs == ys;
    }
};
```

OK, passed 100 tests.

```
int main()
{
    PReverseCancelsReverse revRev;
    revRev.check();
}
```

Falsifiable after 2 tests for input:
0: [-2, 2]

GBM properties

Turning off the “shock” (or vol) gives the drift parameter:

```
class ZeroVolGivesDrift : public Property<ZeroVolPriceGenerator> {  
  
    bool holdsFor(const ZeroVolPriceGenerator& gen) {  
        ...  
    }  
  
    bool accepts(const ZeroVolPriceGenerator& gen) {  
        ...  
    }  
};  
  
int main() {  
    ZeroVolGivesDrift zeroVol;  
    zeroVol.check(100);  
}
```

```

class ZeroVolGivesDrift : public Property<ZeroVolPriceGenerator> {

    bool holdsFor(const ZeroVolPriceGenerator& gen) {
        //Price returns are normally distributed.
        //The *prices* are log normal
        std::vector<double> xs = gen.prices();
        double tot = 0;
        double previous = gen.StartPrice();
        for(const auto & p : xs) {
            tot += (p - previous)/previous;
            previous = p;
        }
        auto average = tot/xs.size();
        return average == gen.Drift(); //or your fav floating point cmp
    }

    bool accepts(const ZeroVolPriceGenerator& gen) {
        return gen.Sims() >= 1 //is 0 ok?
            && gen.Dt() > 0.0
            && std::abs(gen.Drift()) < 20.0;
    }
};

```

```
void generate(size_t n, ZeroVolPriceGenerator & out)
{
    double price;
    double drift;
    double dt;
    int sims;
    generate(n, price);
    generate(n, drift);
    dt = 1;
    generate(n, sims);
    out.reset(price, drift, dt, sims);
}
```

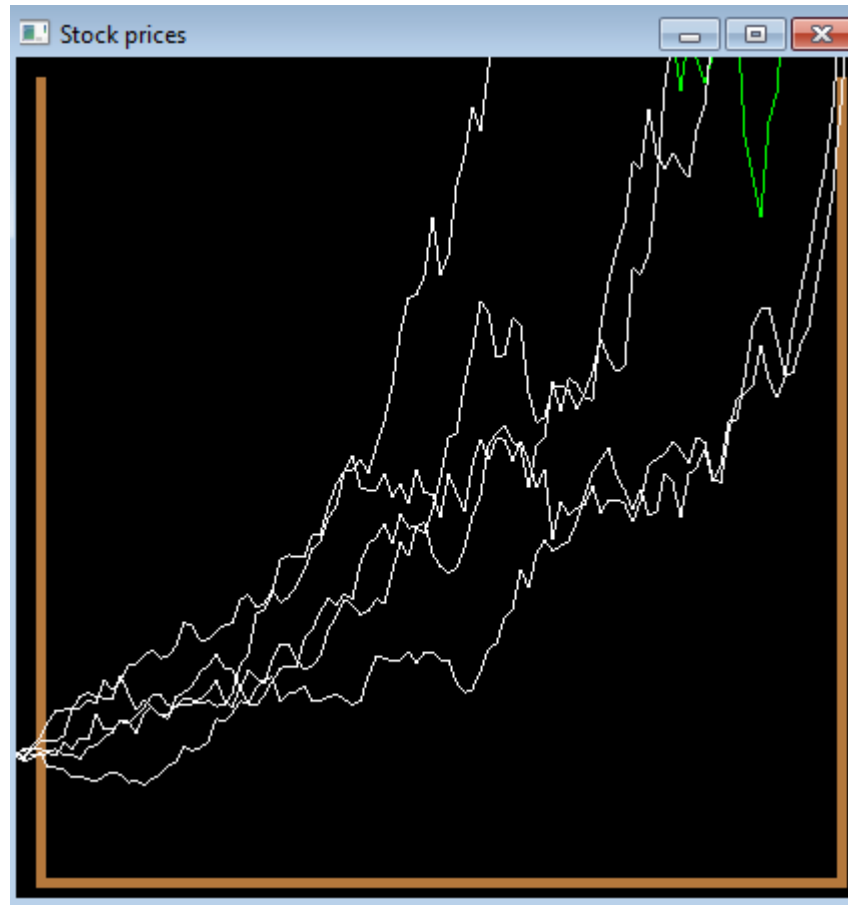
Nice #fail messages

```
std::ostream & operator << (std::ostream & os,  
    const ZeroVolPriceGenerator & gen) {  
    os << "Start price " << gen.StartPrice()  
        << ", drift " << gen.Drift() << ", dt "  
        << gen.Dt() << ", sims " << gen.Sims()  
        << '\n';  
  
    for(const auto & price : gen.prices())  
        os << price << '\n';  
  
    return os;  
}
```

Action...

- (Note to self – do a demo)
 - StockPricePropertyTest.exe
 - *and* stock.bat
 - and variations, e.g stock_no_vol.bat

Ta-da!



But...

- They don't always escape.
- We could just turn the vol off and amp up the drift.
- Or, we could jump...

Jump

- $\Delta S = S(\mu\Delta t + \sigma\sqrt{\Delta t}dW + (J - 1)dN)$
 - J is how much it jumps
 - dN is a Poisson distribution
 - How many jumps in a dt
 - No longer continuous
- <http://demonstrations.wolfram.com/OptionPriceInMertonsJumpDiffusionModel/>
- <http://stackoverflow.com/questions/9870541/using-one-random-engine-for-multi-distributions-in-c11>

One* small change

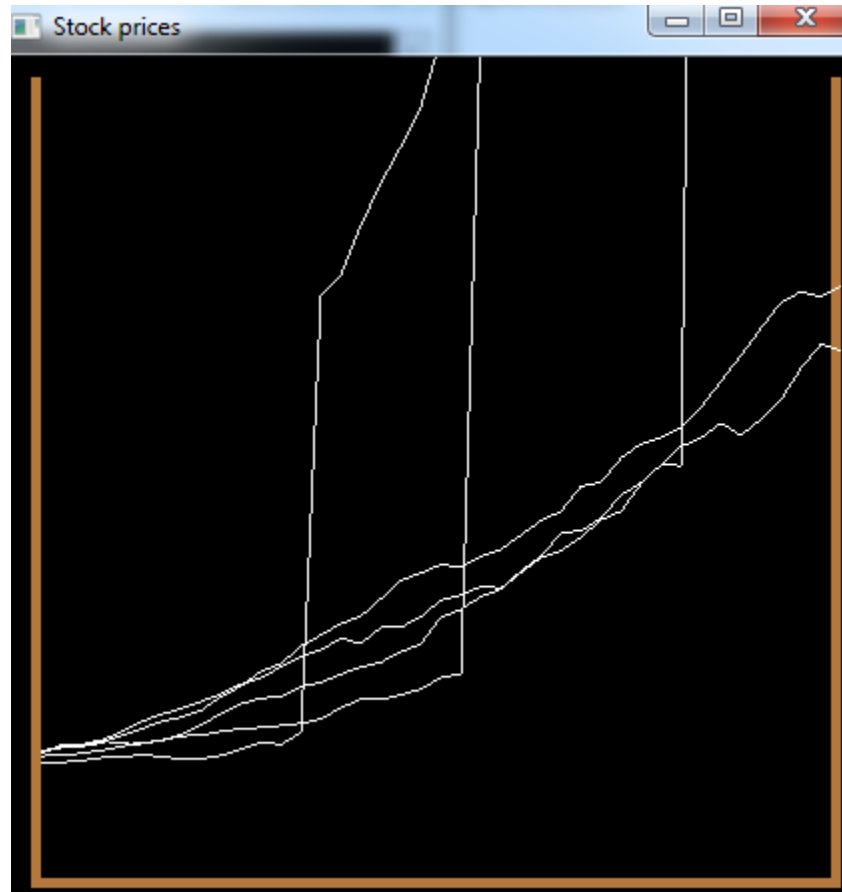
```
double PriceSimulation::update()
{
    double stochastic = normal_dist(engine);
    double dn = poisson_dist(engine);
    double increment =
        drift * dt
        + vol * sqrt(dt) * stochastic
        + jump * dn;
    price += price * increment;
    return price;
}
```

* Alright – two – and the poisson_dist member

Action...

- (Note to self – do a demo)

Ta-Da-Da



Possible extensions

- 3D
- Add a heat source
- Firework algorithm
 - See e.g. “Firework algorithm optimisation” James McCaffery <https://msdn.microsoft.com/en-us/magazine/dn857364.aspx>
 - <http://www.irma-international.org/viewtitle/105168/>
 - <http://www.cil.pku.edu.cn/research/fa/>
- Feel free to try programming your own way out of a paper bag

Let the computer figure it out

- Would be nice if it was a bit quicker
- Time for some machine learning
- Swarm algos
 - We have a swarm of random walkers
 - Want them to move out

It's as easy as A. B. C. ...

Artificial bee colony

- <http://mf.erciyes.edu.tr/abc/publ.htm>
- http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm
- https://www.youtube.com/watch?v=5lz_HnOLBW4



Algorithm

An autocatalytic process:

For a while

Go out

Worker bees

get food from a known food source
and explore nearby

Inactive bees

wait at home

Scout bees

just explore remembering the better food sources

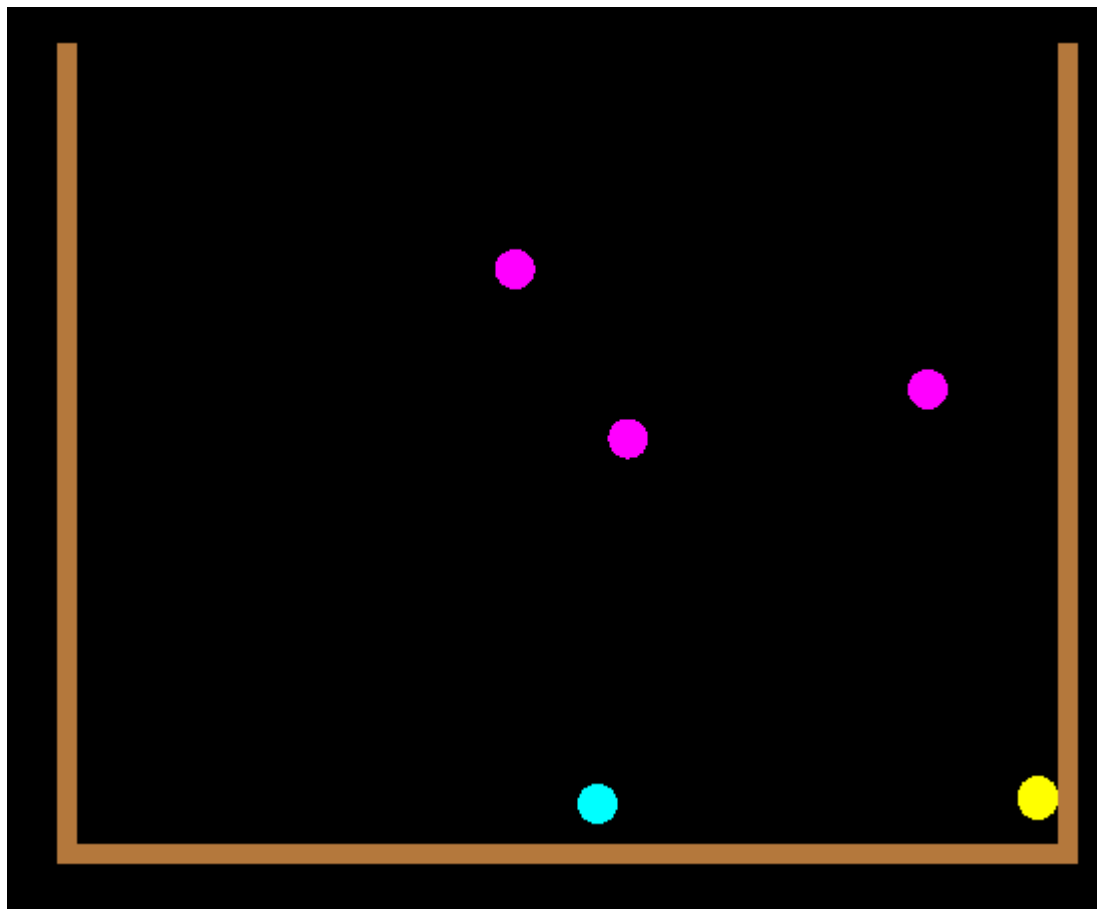
Go Home

Waggle dance

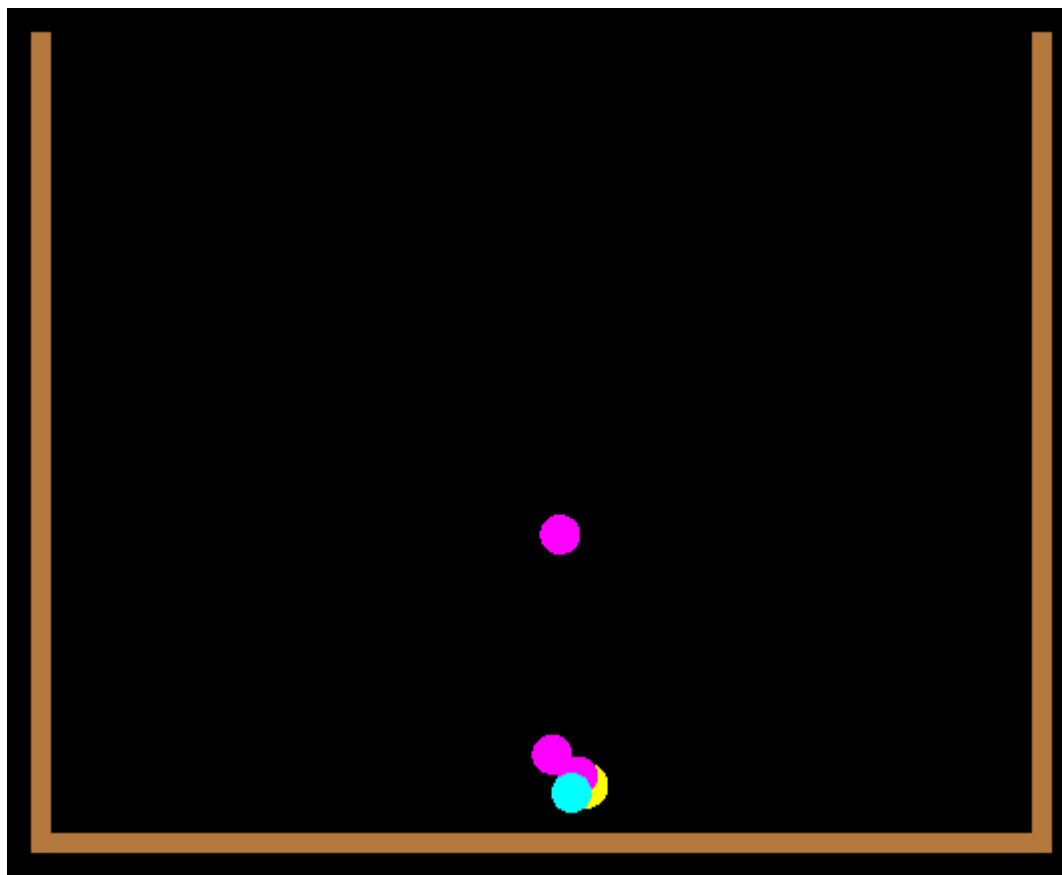
length proportional to food gathered
recruit workers

Maybe swarm

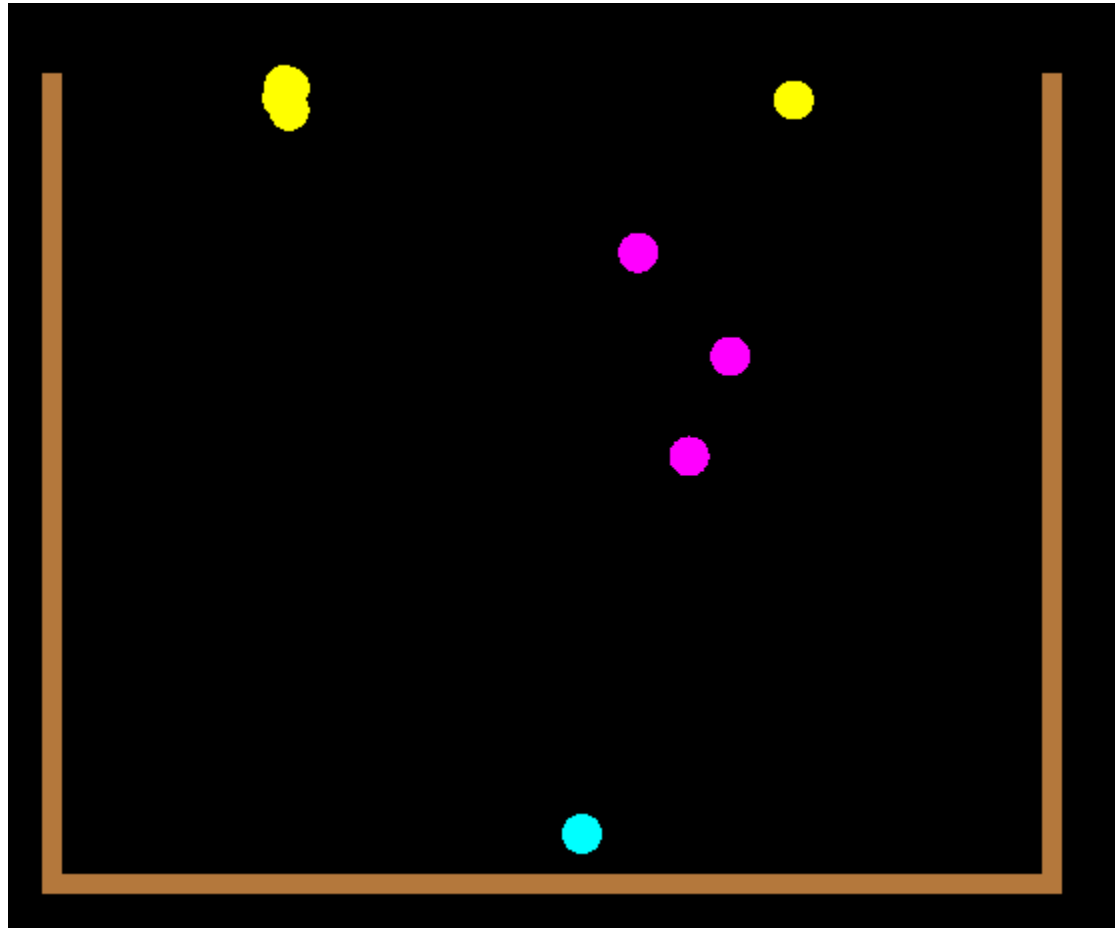
Roles



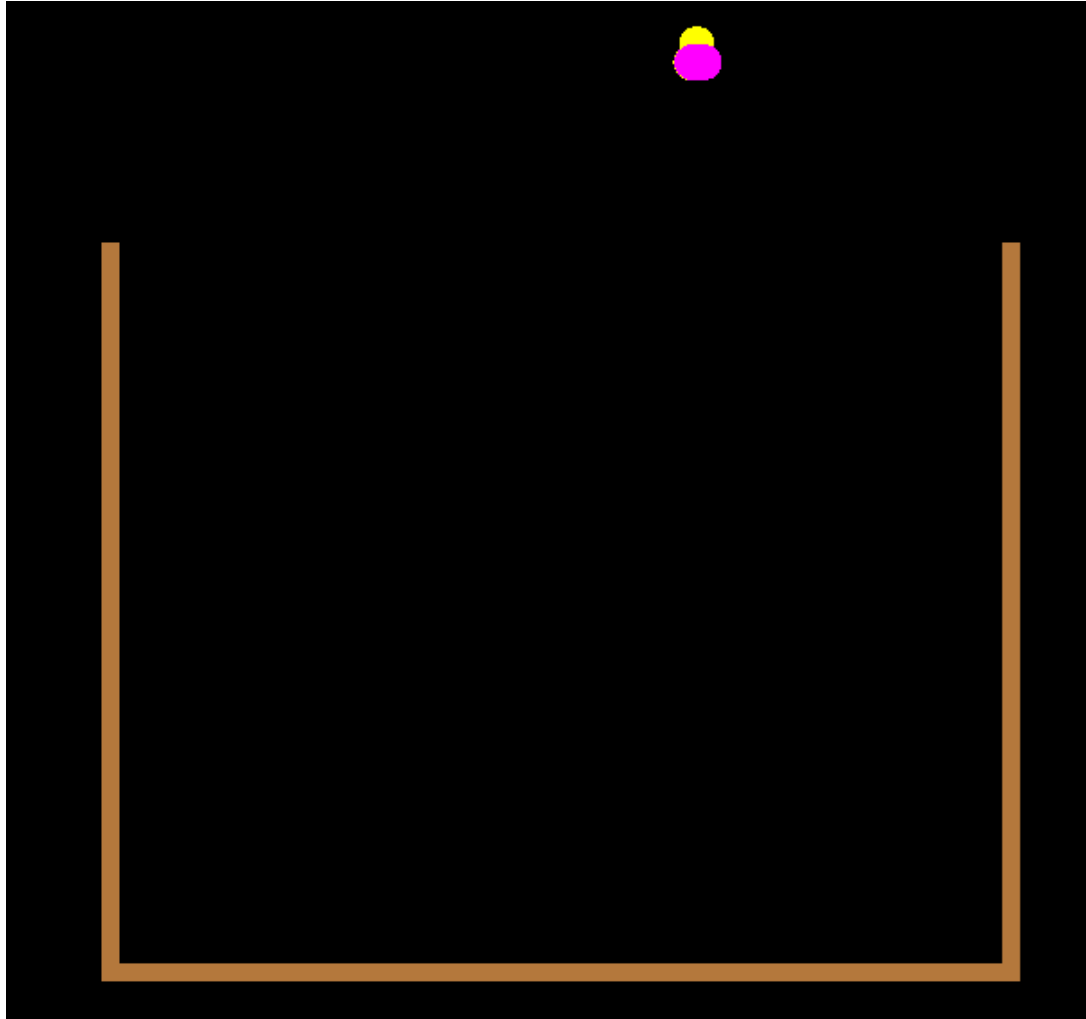
Go home; waggle;



More known food sources



SWARM!



Action...

- (Note to self – do a demo)

Wrap up

- ABC isn't diffusion
 - But built on the 2D Brownian motion
 - And was quicker
- 2D Brownian motion is simple to simulate
 - Provided good $N(0, 1)$ random numbers
- Geometric Brownian motion is also simple to simulate
- Testing properties might be better than lots of hand rolled tests