How to evolve your way out of a paper bag

April 2016

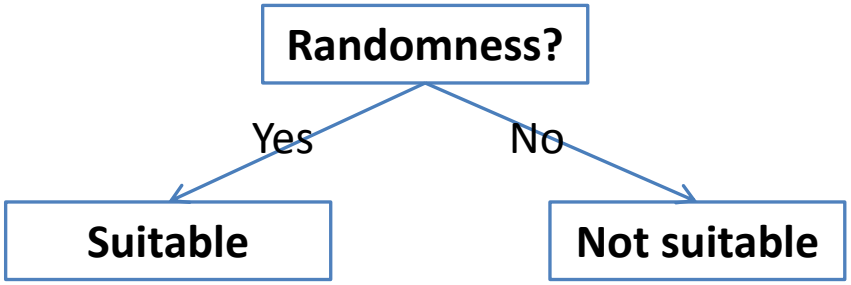
Frances Buontempo
@fbuontempo
frances.buontempo@gmail.com
overload@accu.org

https://github.com/doctorlove/paperbag

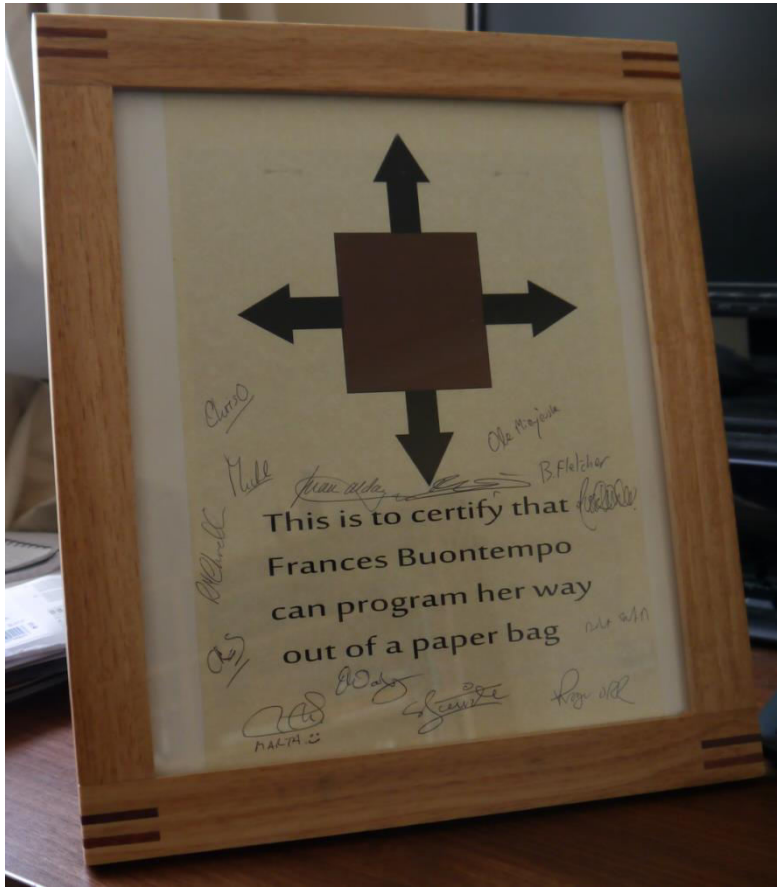
Objectives

- Evolution
- Genetic algorithms
- Artificial life
- Testing – property based, fuzz, mutation, …
- Automate EVERYTHING – even the code gen?

Background

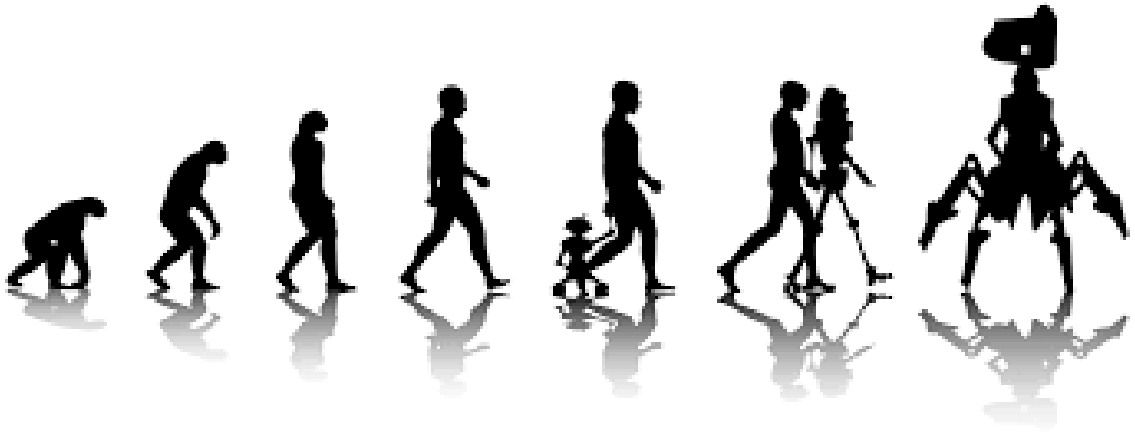


C4.5 see
https://www.rulequest.com/



http://lmgtfy.com/?q=frances+buontempo+paper+bag

Today - evolution



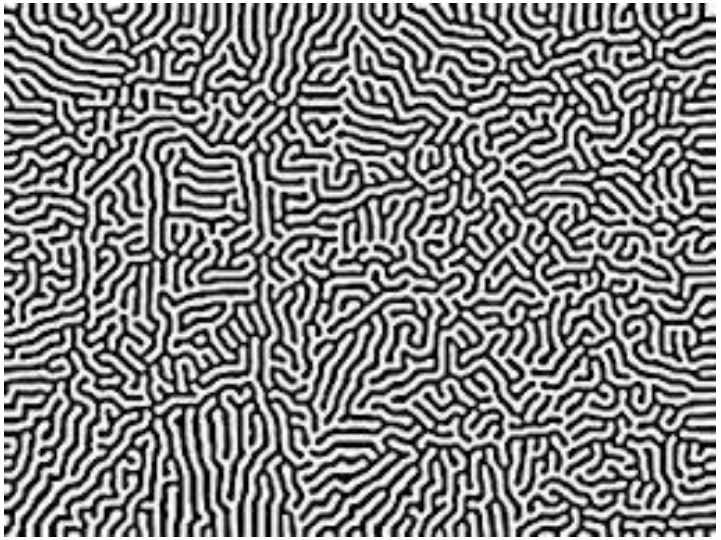
mutation

fitness

Genetic algorithms

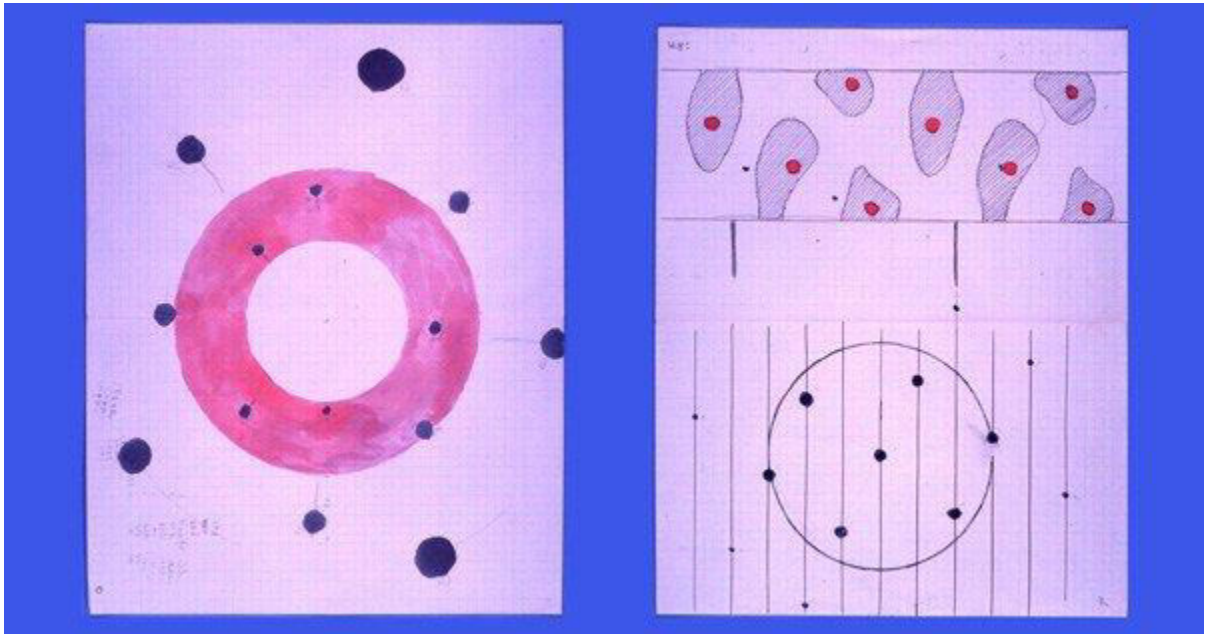
- AI
  - Never forget the Turing Test
  - How the zebra got its stripes
    - Reaction-diffusion, morphogens
    - “Turing systems are completely self-contained, self-starting and self-organising. “
- GA
- Ballistics – 2 parameters
- Possible extensions

Aside - zebras



- https://en.wikipedia.org/wiki/Reaction%E2%80%93diffusion_system

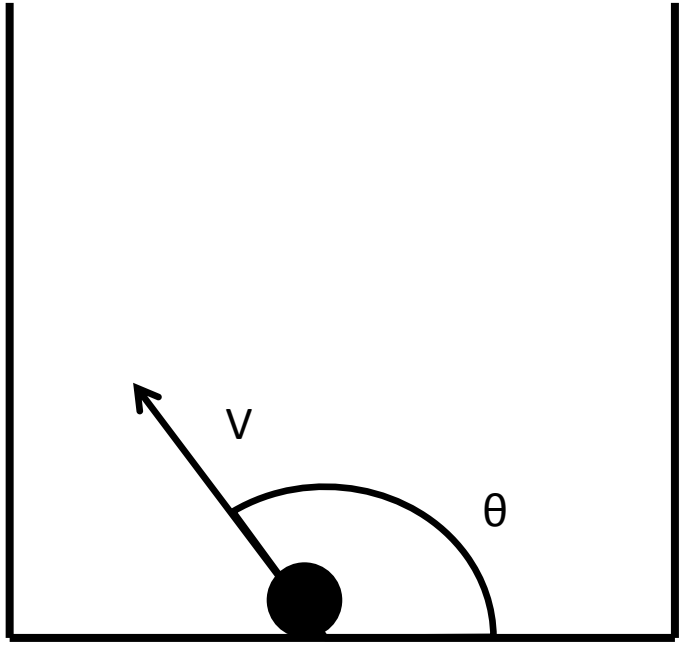
Some of Turing's drawings



http://www.turingarchive.org/browse.php/K/3

Ballistics

Equation



$$x = vt\cos\theta$$

$$y = vt\sin\theta - \frac{1}{2}gt^2$$

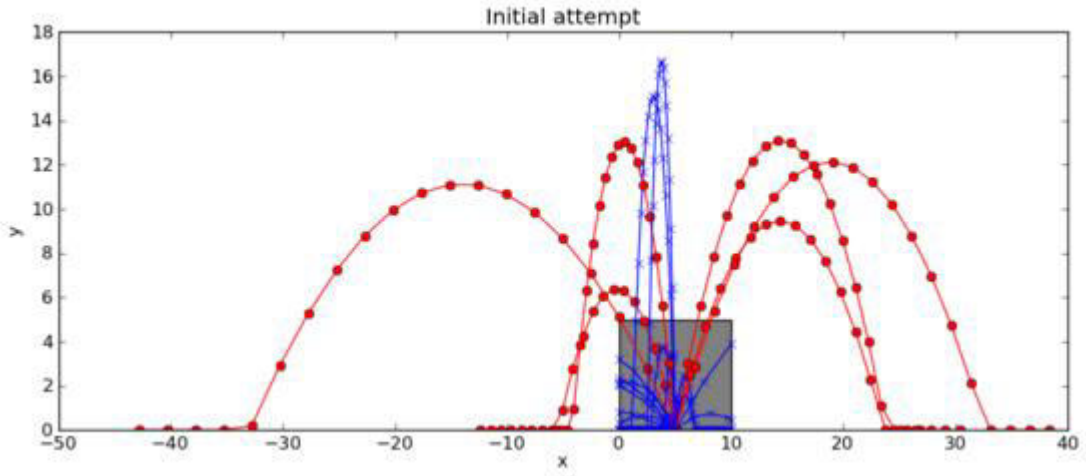
- *Overload*, 21(118):7–9, December 2013
  - http://accu.org/index.php/journals/1821

Algorithm

- Random initial generation of n items
  - Pairs of (angle, velocity)
  - How many? (More than 1)
- For a while
  - Launch
  - Crossover – some of the “best”
  - Mutate

Random start

- n pairs of (angle, velocity)
  - each within a range
- run the trial
  - launch and observe



Initialise

```
def init_random_generation(items):
    generation = []
    for i in range(items):
        theta = random.uniform(0, 180) * math.pi/180
        v = random.uniform(2, 20)
        generation.append((theta, v))
    return generation
```

Next generation

- From the current generation
- Breed a new generation
  - None live for more than one generation
  - Could try keeping some of the best instead
- How?
  - Select parents (fitness)
  - Crossover
  - Maybe mutate

Which?

- Fitness function
  - Best, better?
  - Any that escaped?
  - What if none escape?
  - What if some nearly escape?
- When do they die?
- Should the population size stay stable?

Selection schemes

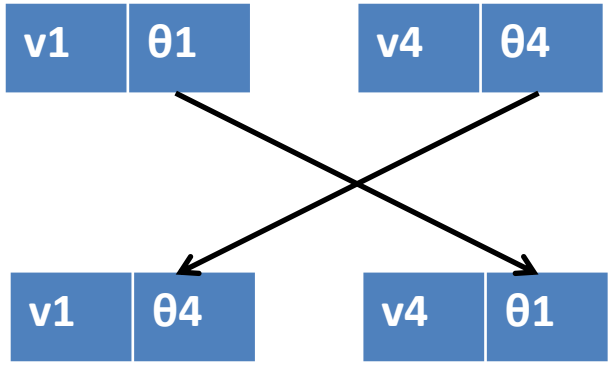
- How do you select a selection scheme?
  - ranking selection (Baker 1985)
    - linear ranking
  - proportionate reproduction
    - roulette wheel (DeJong 1975)
  - tournament (Brindle 1981)
    - binary, best of three,…
- Do we keep the best ones?
  - “steady-state” birth and death via ranking

(bad) choice ga1.py

```
def get_choices(gen, results):
    choices = [(gen [i][0], gen [i][1]) \
       for i in range(len(gen)) if escaped(results[i])]
    return choices


…
choices = get_choices(generation, results)
if len(choices) == 0:
    return init_random_generation(items)
```

What happens if none escape?

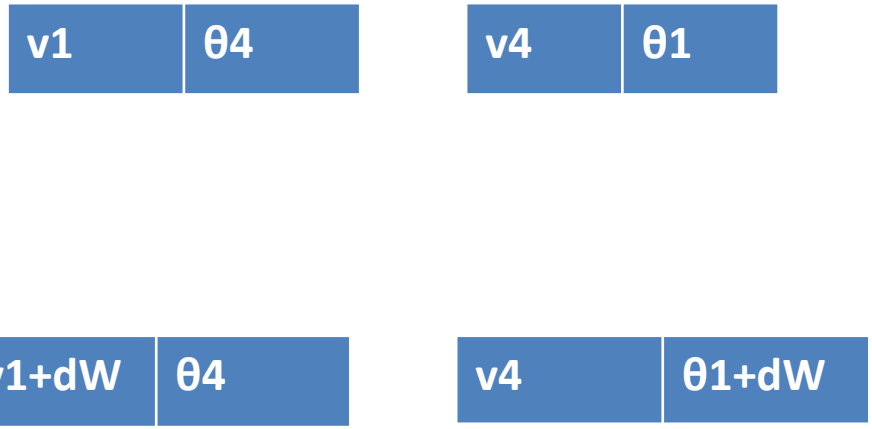
Crossover



Spawn

```
def crossover(gen, res):
    choices = get_choices(generation, results)
    next_generation = []
    for i in range(0, len(generation)):
        mum = generation[choose(choices)]
        dad = generation[choose(choices)]
        t = (mum[0], dad[1])
        next_generation.append(t)
    return next_generation
```

Mutation



Godzilla

```
def mutate(generation):
    for i in range(len(generation)):
        (theta, v) = generation[i]
        if random.random() < 0.1:
            theta += random.uniform(-10, 10) \
                     * math.pi/180
        if random.random() < 0.1:
            v *= random.uniform(0.9, 1.1)
        generation[i] = (theta, v)
```

Maybe Gaussian is better

Pulling it together

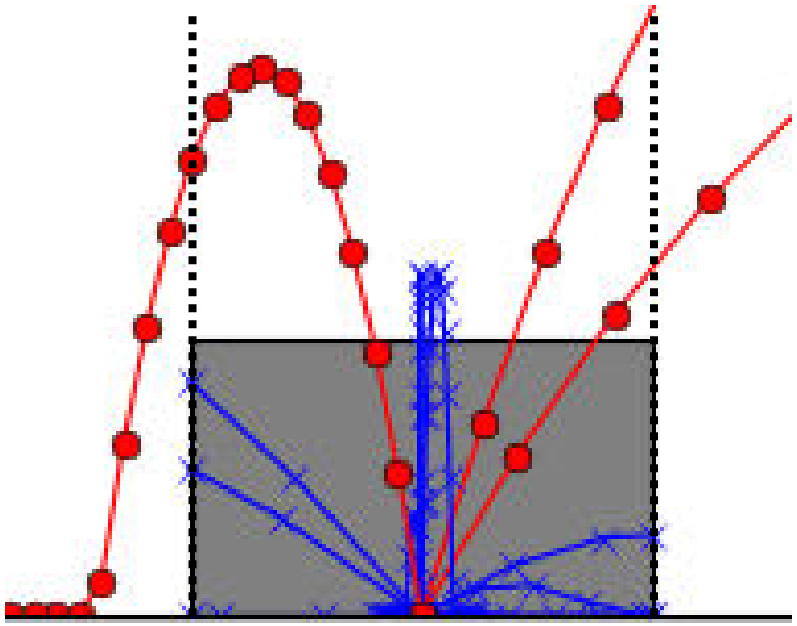
```
#epochs = 10, items = 12, height = 5, width = 10

gen = init_random_generation(items)

results = launch(gen, height, width) #or do the maths

for i in range(1, epochs):
    gen = crossover(gen, results, height, width)
    mutate(gen)
    results = launch(gen, height, width)
```

Action!

- (Note to self – do a demo)

Fitness again

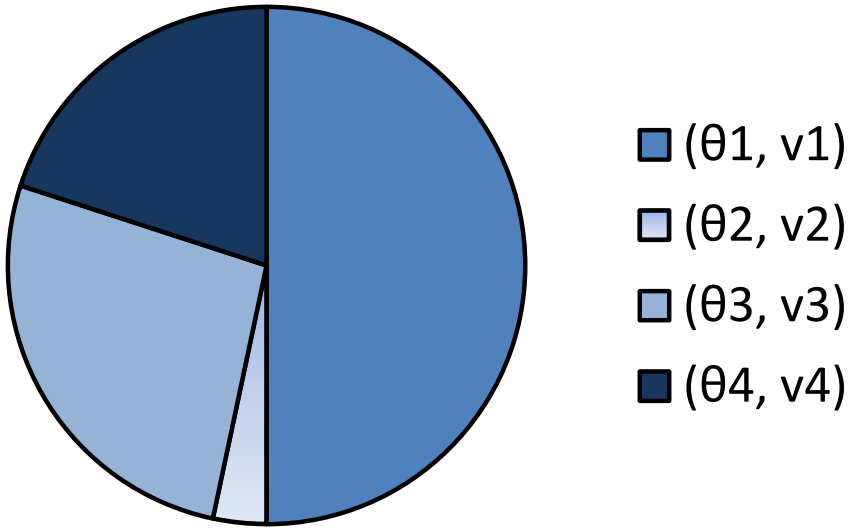
- Something either ends in or out of the bag
- But some fail cases are less bad than others



- 3 escape
- 2 on left get “close”
- Could “close” mean height (at edge of bag)?
  - Fitness = height

Roulette wheel – more likely to pick better chromosomes

(θ, v)	fitness	Σ(fitness)
(θ1, v1)	15	15
(θ2, v2)	1	16
(θ3, v3)	8	24
(θ4, v4)	6	30



Chose a number in range [0, 30) and select the corresponding chromosome pair (θi, vi)

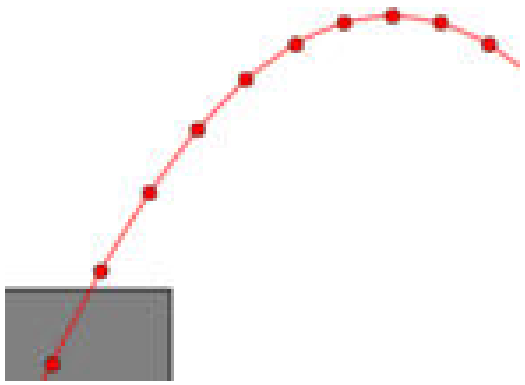
(better) choice ga2.py

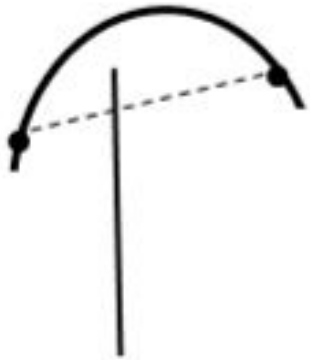
```
def cumulative_probabilities(results):
    cp = []
    total = 0
    for res in results:
        total += res[1] # height
        cp.append(total)
    return cp


…
#choices = cumulative_probabilities(results)
def choose(choices):
    p = random.uniform(0, choices[-1])
    for i in range(len(choices)):
        if choices[i] >= p:
            return i
    return i
```

Is this a bit untidy?

Observation

It’s quicker to do the maths and calculate where it is at the bag width or if it goes out the top, than to run the “experiment” and interpolate between points at the bag width





Action…

- (Note to self – do a demo)
  - Saw ga1.py
    - “Escaped” fitness function - slower
  - ga2.py
    - Better fitness function - quicker
  - ga.py
    - No interpolation
    - Better fitness function
  - Worst names in the world!

Why?

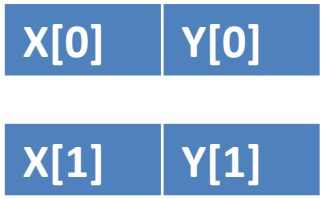
- Forces
  - Recombination (crossover) + mutation = diversity
  - Selection = quality
- Potentially explore all the space
  - Avoid local minima/maxima
- See e.g. A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer,
  First edition, 2003, ISBN 3-540-40184-9, Corrected 2nd printing, 2007, ISBN: 978-3-540-40184-1

Magic numbers

- Lots of magic numbers
  - Bag size, gen size, epochs
- Including g = 9.81
  - Could we go into orbit around the bag instead?
  - Should it be circular?
  - Why not let it choose the best gravity
    - and the number in each generation
    - and how many epochs
    - and how much mutation
  - Metaheuristics

Shape of GA

- Fixed length vector of parameters



- More than 2 – where to split?

- Can we have varying number of parameters or a tree structure?

Further reading

- J. Holland,
  - *Adaptation in Natural and Artificial Systems Uni Of Michigan Press 1975*
  - "Genetic Algorithms," Scientific American, Vol. 267, No. 1, 1992, pp. 66-72
- D. Goldberg
  - *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer 1989
- Some coded for you
  - WEKA*, http://pyevolve.sourceforge.net/ , **lots** of C++ *
  - .Net: http://johnnewcombe.net/gaf/
- Distributed evolutionary algos in Python https://github.com/deap/deap
- Koza: http://www.genetic-programming.com/

*http://www.omicsonline.org/open-access/applying-weka-towards-machine-learning-with-genetic-algorithm-and-backpropagation-neural-networks-2153-0602.1000157.pdf

*Gagné, Christian, and Marc Parizeau. "Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computation." GECCO Late Breaking Papers. 2002. http://w3.gel.ulaval.ca/~cgagne/pubs/lbp-gecco02.pdf

Artificial life

- Man-made systems
- Behavioural characteristics
- Software, hardware, wetware
- Resources
  - http://www.mitpressjournals.org/loi/artl
  - http://www.alife.org/
  - http://sig.sigevo.org/index.html/tiki-index.php
    - E.g. GECCO

History lesson

- Von Neumann introduced idea of
  - “self-reproducing cellular automata: computer programs capable of making copies of themselves”
  - “contain a set of instructions that are then copied to the offspring”
  - Maybe in the 1950s, maybe with Stanislaw Ulam
- c.f. DNA
- Holland picked up the idea
- A link with nonlinear feedback shift registers and cryptography

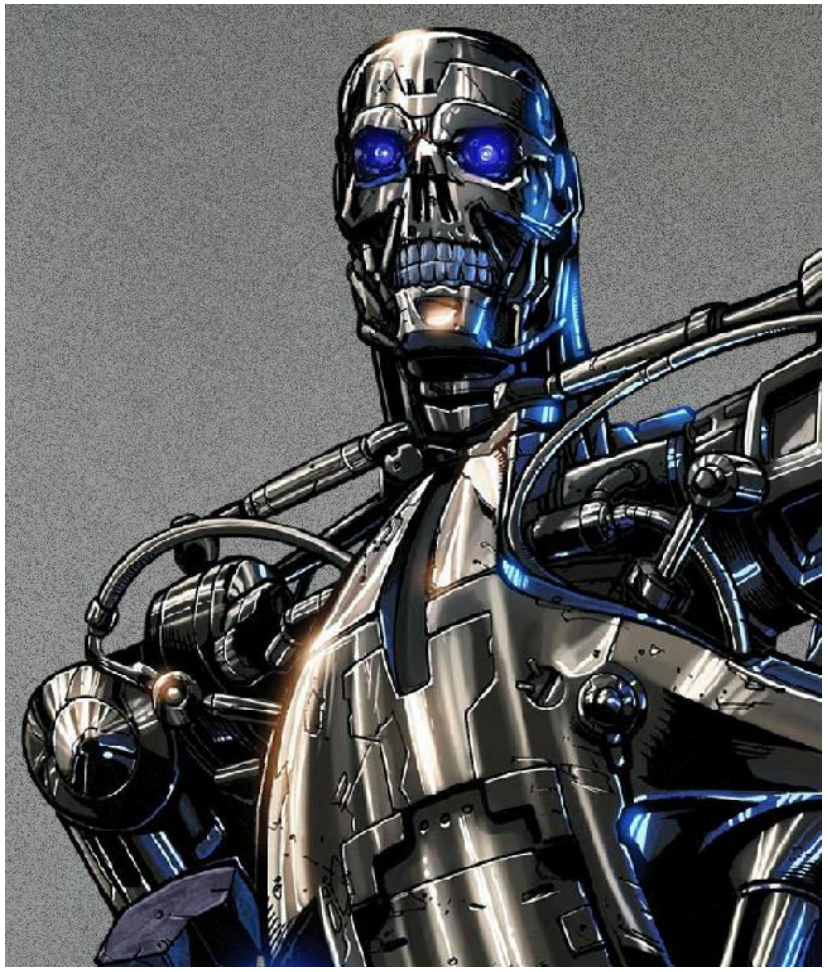
Barricelli

- 1953
- IAS machine, Princeton
  - Weather forecasting/Los Alamos nuclear weapons
  - He “ finagled time on the computer to model the origins and evolution of life.”
    - http://nautil.us/issue/14/mutation/meet-the-father-of-digital-life
  - Made artificial universes, using randomly shuffled playing cards for random inputs
- “Numerical organisms”

The rise of the machines

Turing's hope that "[...] machines will eventually compete with men in all purely intellectual fields" is far from accomplished. …
When Alife began to lose its momentum several years ago, **biologically inspired** (or nature-inspired) computer science became a buzzword and new ultimate design paradigm, whose broadly defined mission is—not unlike Alife—was to mimic instead of copy nature. The field of biologically inspired computer science is generally more concerned with **solving real problems** and **building more powerful machines**, unlike the Alife mantra of "discovering how life works by building it."

Biologically uninspired computer science, Teuscher, Communications of the ACM 49.11 pp27-29



Machines or people?

- Or security?

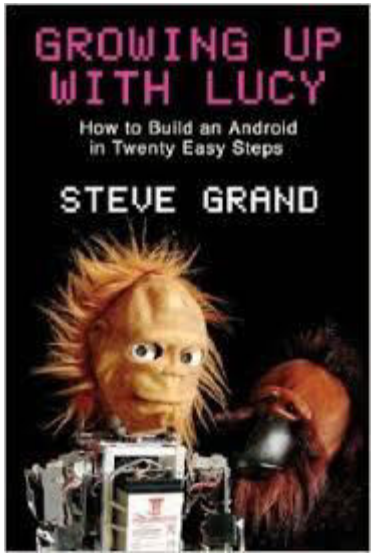
What's the point?

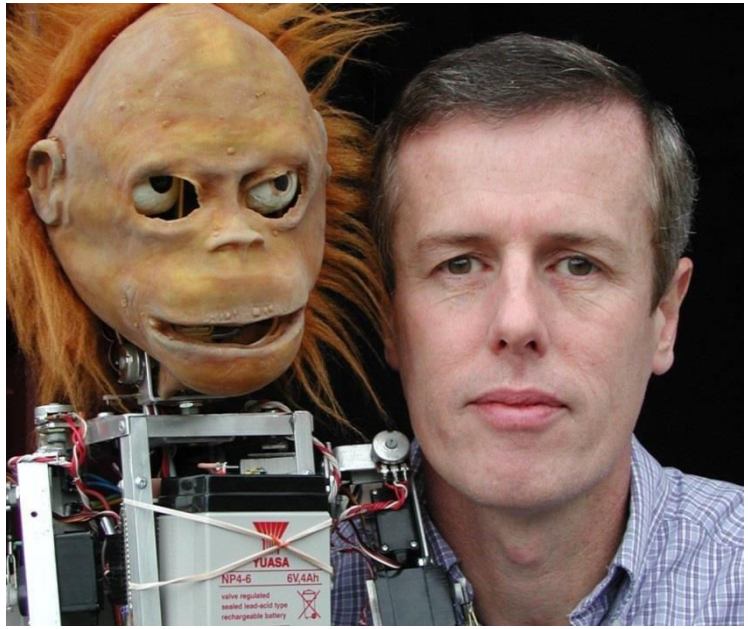
- "Discovering the laws of urbanisation" Filippo Simini, Charlotte James 2015
- Computer generated art, music…
- Evolvable hardware
- Optimisation problems esp multi-objective
- Scheduling
- Pretty pictures, fun, …

Aside

- Creatures (Norns, etc) – Steve Grand
  - A neural network (but you interact)?
  - https://en.wikipedia.org/wiki/Artificial_life
  - http://creatures.wikia.com/wiki/Creatures

Steve Grand

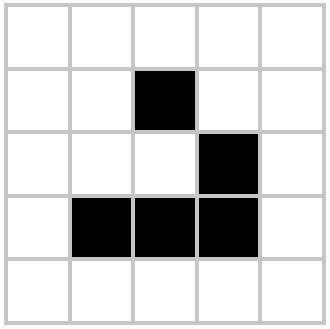




https://stevegrand.wordpress.com/106-2/

Conway’s Game of Life

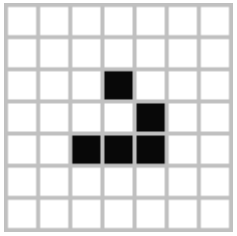
The fantastic combinations of John Conway's new solitaire game "life“
Martin Gardner Scientific American 223 (October 1970): 120-123.
http://www.ibiblio.org/lifepatterns/october1970.html



Rules:

1. Fewer than two live neighbours dies
2. Two or 3 live neighbours lives
3. More than three neighbours dies
4. Dead cell with exactly 3 live neighbours lives

- http://catagolue.appspot.com/
- @conwaylife



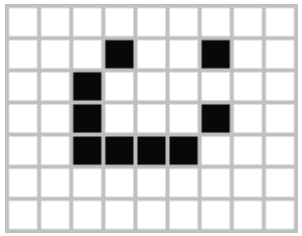
Train of thought

- Michael Feathers on the future
  - reactive patterns
  - microservices
  - “collection pipeline programming”
  - GoL in APL

life←{↑1 ωV.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1∘.⌽⊂ω}

https://en.wikipedia.org/wiki/APL_(programming_language)#Game_of_Life

Known patterns



- Still life
  - Stable shapes
- Oscillators
  - Cycle back to starting point in n steps = period
  - Some with >> 100
  - None found (yet) with 19, 23, 38 and 41
  - See http://www.conwaylife.com/wiki/Oscillator
- Spaceship
  - Oscillator like but moves

Pictures in C++

```
#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();
    }
}
```

Finally, some code

```
class World
{
public:
    //Constructors
    bool Alive(size_t x, size_t y) const
    {
        return state[y*max_x + x];
    }
    bool Update();
private:
    const size_t max_x;
    const size_t max_y;
    std::vector<bool> state;//evil
};
```

Evil vector bool

- https://isocpp.org/blog/2012/11/on-vectorbool
- Why is vector bool so evil anyway?

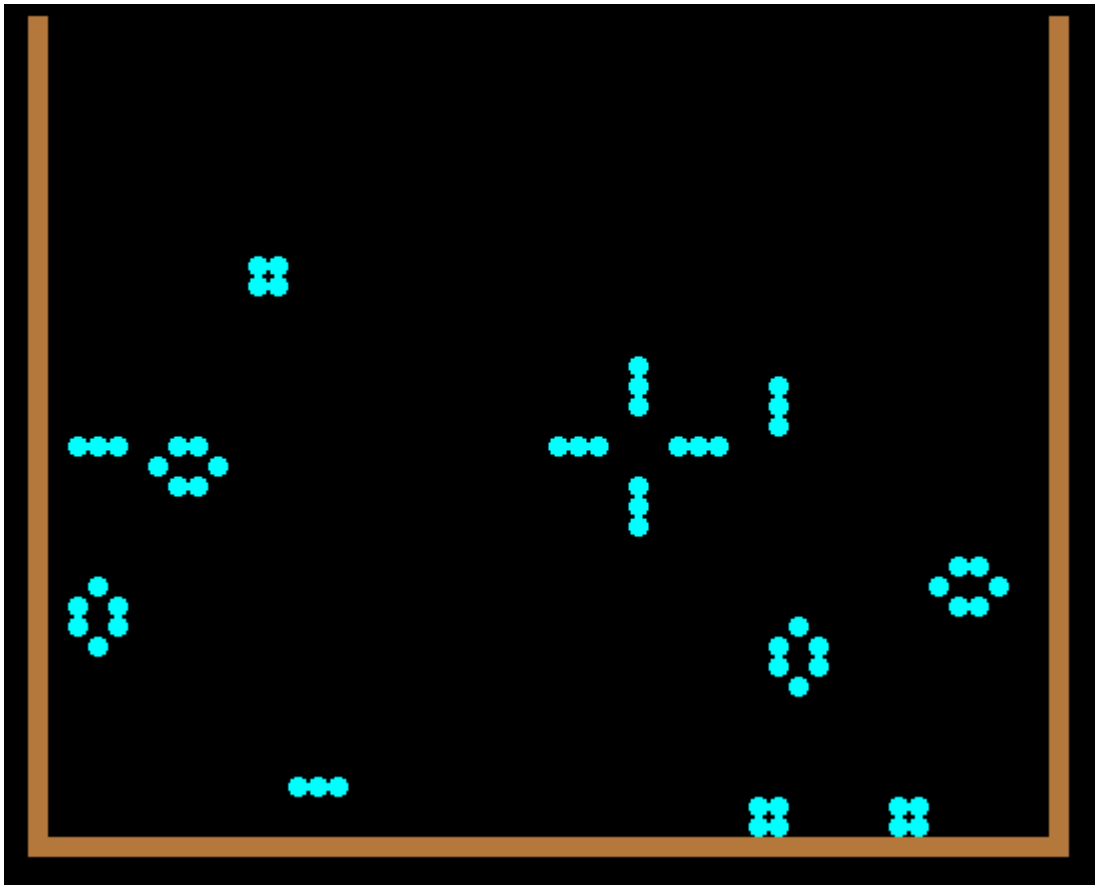
If you thought that was evil…

```
void walk_neighbours(size_t x, size_t y, size_t max_x, size_t max_y,
                     std::function<void(size_t, size_t)> action) {
    if(y>0) {
        if(x>0)
            action(x-1,y-1);
        action(x,y-1);
        if(x<max_x-1)
            action(x+1,y-1);
    }
    if(x>0)
        action(x-1,y);
    if(x<max_x-1)
        action(x+1,y);
    if(y<max_y-1) {
        if(x>0)
            action(x-1,y+1);
        action(x,y+1);
        if(x<max_x-1)
            action(x+1,y+1);
    }
}
```

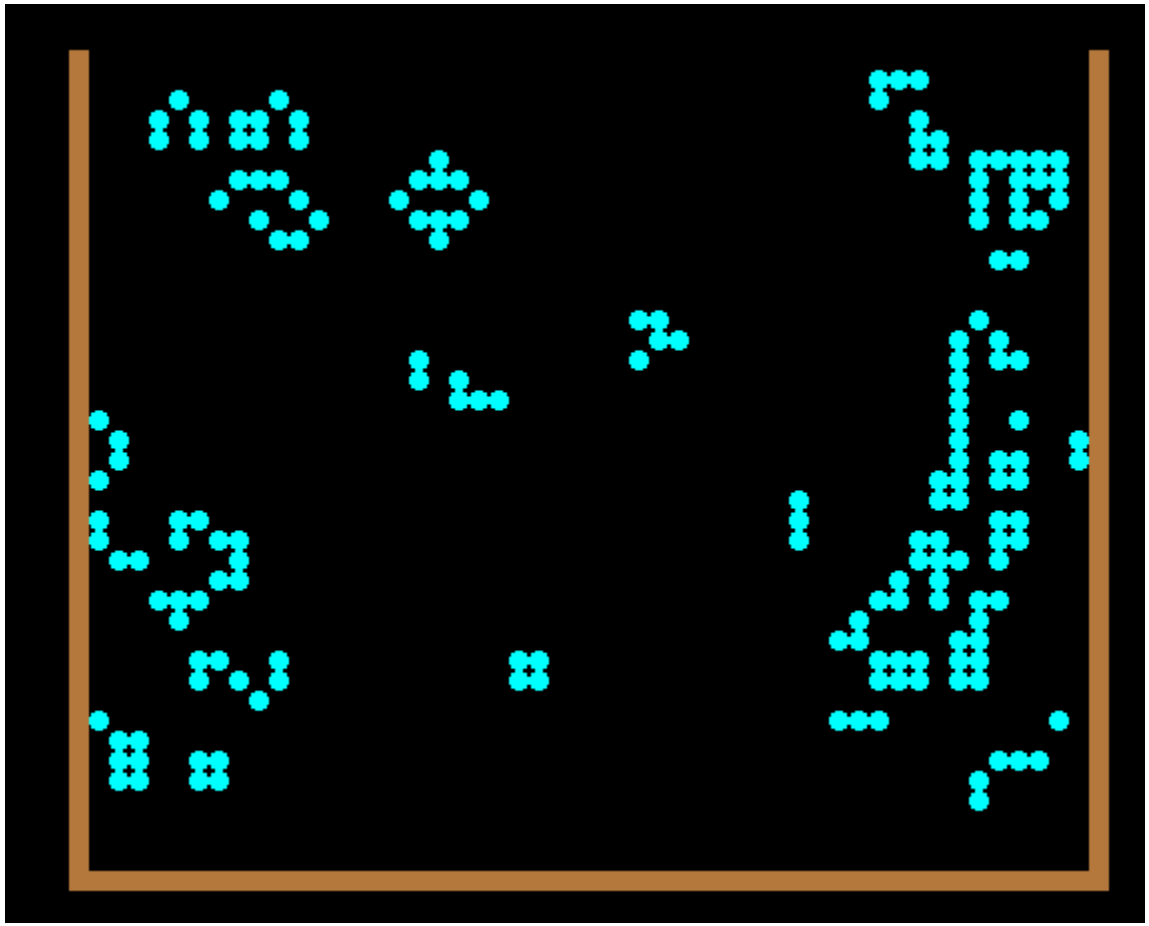
A slight oversight

- Edges make it all go wrong
- Allow wrapping?
- BUT there’s no way out…
- (note to self – demo)

Stabilised



Wrapping – spot the glider

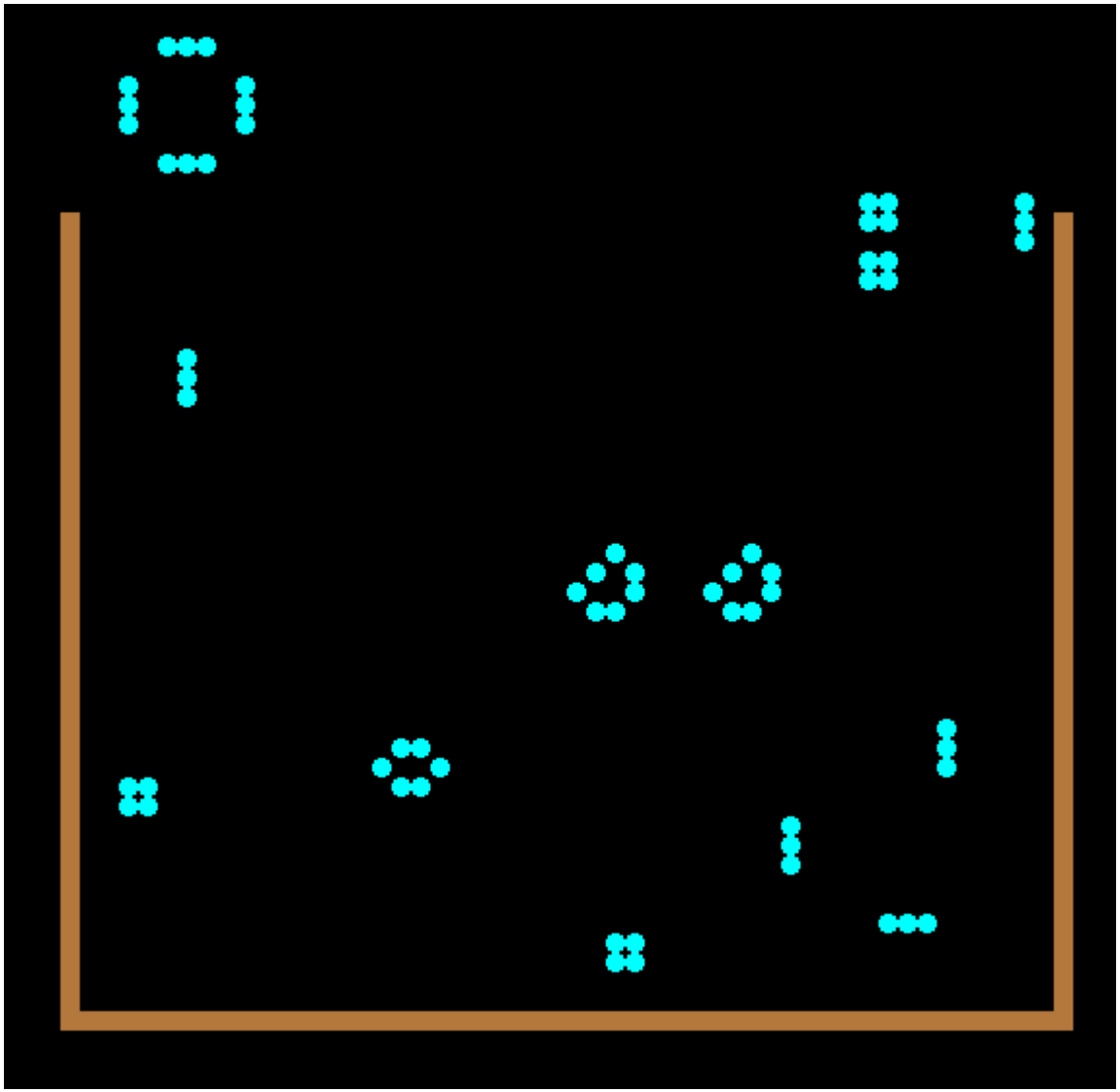


Let me out of here

- Doesn’t stabilise as quickly if it wraps
- But the top joins the bottom so none escape
- Make the world taller than the bag
- Some may not escape…

Above the bag

…but not all of them



Action…

- (Note to self – do a demo)
  - Evolution.exe
  - Evolution.exe wrap
  - Evolution.exe out

Mutate the rules

- GA the number of neighbours
  - 0 to 8 immediate ones
  - Each is either dead or alive
  - Lookup table?
  - Fill the bag – might bust the edges ☺
- Left as an exercise for the listener

Other Cellular Automata

- Langton's ant
  - https://en.wikipedia.org/wiki/Langton%27s_ant
- Worms…
- "Rock, paper, scissors" (warning – unpleasant colours) [red eats blue, green eats red, and blue eats green] https://www.youtube.com/watch?v=M4cV0nCIZoc
- And many more

Testing, testing

- How do we test our code?
- Manually
- Automatically
  - Hand rolled unit tests
  - Integration tests
  - Static analysis
  - LLVM fuzzers
  - Mutation testing
  - Property based testing
  - …

#ACCU2016

- Talking of testing
- Thurs?
  - Mutation Testing in Python, Austin Bingham
- Fri?
  - Property Based Testing Hands-on in Haskell or Javascript, Willem van der Ende + Marc Evers

Property based testing

- “Generating test cases so you don’t have to”
- State properties that should hold
  - it tries to find cases where they don’t
- Roots in Haskell: QuickCheck
- C++
  - RapidCheck
  - https://github.com/emil-e/rapidcheck
  - https://labs.spotify.com/2015/06/25/rapid-check/
- Python
  - Hypothesis
  - http://hypothesis.readthedocs.org/en/latest/

Mutation testing

- “Quis custodiet ipsos custodes?”
- How good are my tests?
- High coverage =>
  - No missing checks
  - Loopholes
  - Edge cases cause trouble
  - Combinations
    - (only in prod of course)

What is it doing?

- Write tests and run them
- Make small changes (mutants) to code and re-run tests
- Mutants usually go against the specifications so tests should fail
  - E.g. replace > with <, ! with !!, …
- If tests still pass => problems
- “ The mutant is *killed* if at least one of the unit test cases has raised an assertion ”

https://miketeo.net/wp/index.php/projects/python-mutant-testing-pymutester

Hang on….

- Should changing code ALWAYS make a test fail?
- TDD
  - *Write a failing test*
  - *Get that test to pass*
  - *Refactor the code and the tests still pass*

Do it!

- https://github.com/miketeo/PyMuTester
- Point at my ballistic tests

```
from mutester.nose_main import main
if __name__ == '__main__':
    main()
```

Kill all mutants

>python pymurun.py --mutant-path .

Mutant Test Results

Total: 46

Alive: 0 (0.0%) Killed: 37 (80.4%) Unreachable: 9 (19.6%)

>grep " Mutant" pymuout

*** IFNOT-1... Mutant killed

*** IFNOT-2... Mutant killed

*** SKIPLOOP-1... Mutant killed

*** IFNOT-1... Mutant killed

*** IFNOT-2... Mutant killed

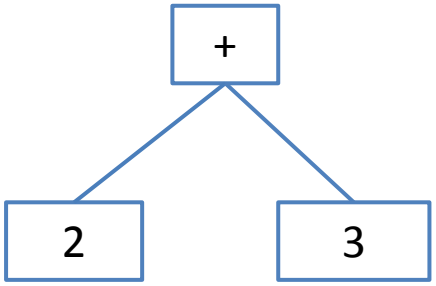
*** IFNOT-1... Mutant killed

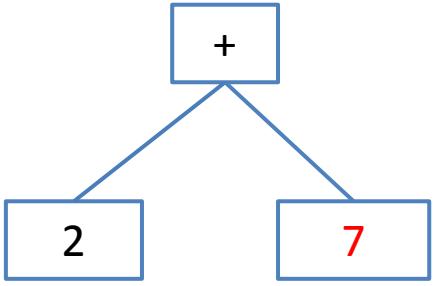
*** IFNOT-2... Mutant killed

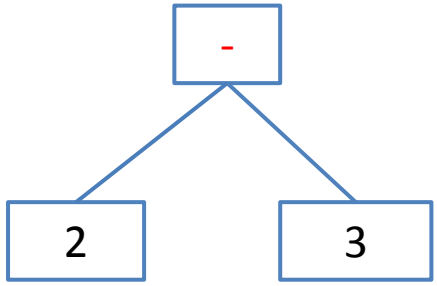
*** SKIPLOOP-1... Mutant not reached

*** IFNOT-1... Mutant killed …

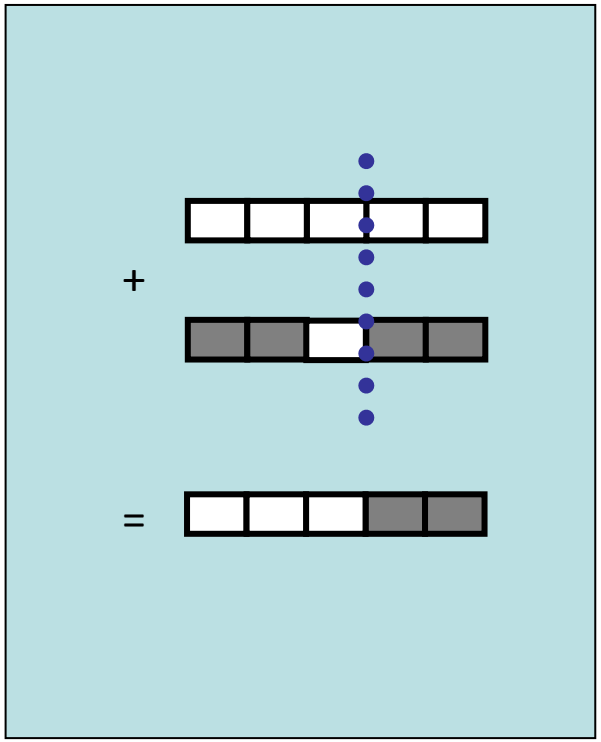
Use the AST

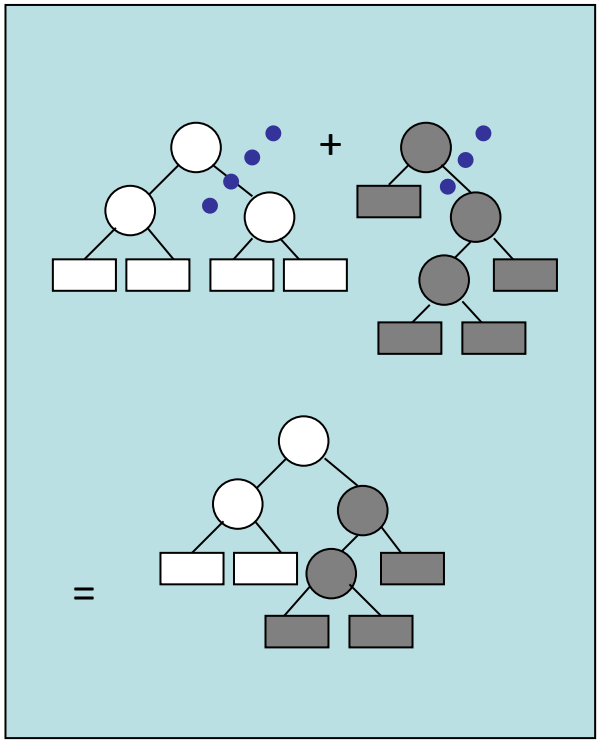






Tree crossover?



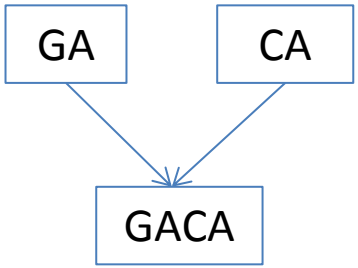


Genetic Programming

- Genetic algorithms operated on an array
- Mutation testing can use an abstract syntax tree
- GA -> GP
  - Crossover and mutate *trees*
  - http://www.genetic-programming.org/
- Search based software engineering (SBSE)
  - Dr Chris Simons @chrislsimons
  - “Metrics are not enough”
    - http://people.uwe.ac.uk/Pages/person.aspx?accountname=campus%5Ccl-simons

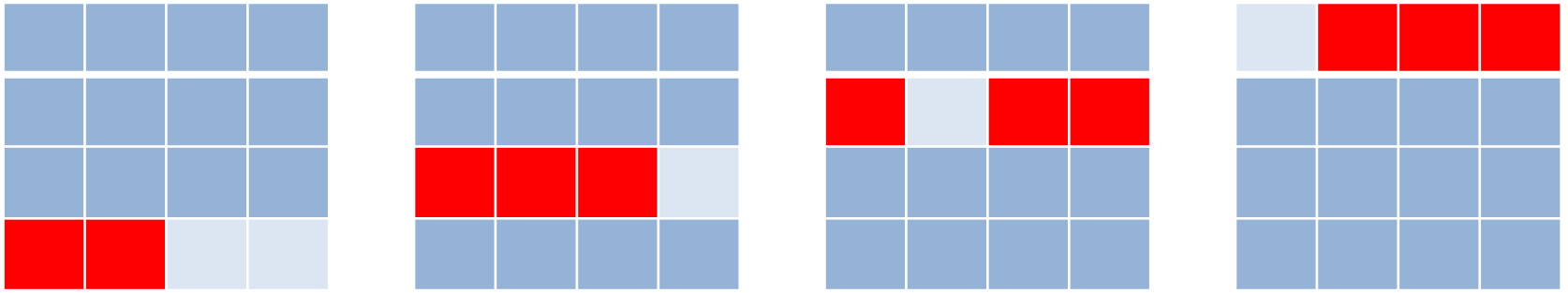
Back to paper bags

- Looked at GA
- Looked at CA
- They didn't all escape
- …. So ….



Very simple cellular automaton

- What if we just go up one row at a time?



GA a CA?

- Evolve some rules or starting cells
- Using GA
- What's the best starting cell?
  - For rule "flip bits"
    - [1, 1, 0, 0] -> [0, 0, 1, 1]
  - For rule "stay same"
    - [_, _, _, _] -> [_, _, _, _]
- What's the best rule for a given cell?

Algo

- Generate starting cells randomly [1, 0, 0, 1]
- For a while
  - Generate some rules
    - cells pattern -> cells pattern
    - [1, 0, 0, 1] -> [1, 0, 1, 0]
    - Note – lazy generation is quicker!
  - Fitness = how many survive to the top
  - Crossover, mutate
  - Report best each time

Initial cell patterns

- Choose cell size
- Each cell is on or off
  - [1, 0, 0, 1]
- Possible extensions
  - Colour
- Do we want uniform distributions?
  - Permutations of a half on/off?
  - Most alive? Most dead?...

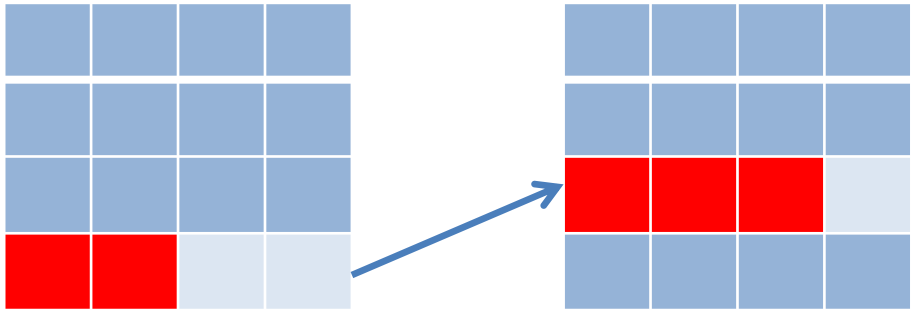
Generate a cell

```
typedef std::vector<bool> Cells;

Cells CellGenerator::generate() {
  Cells cells;
  for(size_t i = 0; i < cell_size; ++i) {
    cells.push_back(uniform_dist(gen) == 0);
  }
  return cells;
}
```

More likely to die

```
#include <iostream>
#include <random>
int main() {
    std::random_device device;  std::mt19937 gen(device());
    std::bernoulli_distribution coin_flip(0.2);

    for (int trials = 0; trials < 100; ++trials) {
        std::cout << trials << ',';
        for (int i = 0; i < 4; ++i) {
            bool outcome = coin_flip(gen);
            std::cout << std::boolalpha << outcome << ',';
        }
        std::cout << '\n';
    }
}
```

What's a rule?



```
class Rule
{
public:
   virtual Cells operator()
      (const Cells & cells) const = 0;
};
```

Generating a rule

- For a given cell, we want to return another cell
- Randomly
- Here's a function we made earlier:

```
CellGenerator::generate()
```

How many rules?

- There are lots - maybe 2^{n^n}?
- Do we need good initial coverage?
  - Or will mutation find missing places?
- What starting cell(s) should we use?
- How do we represent them
  - Especially so we can do “crossover”
  - Input -> output
    - [1, 0, 0, 1] -> [1, 0, 1, 0]

Generating a rule JIT

```
Cells
JitRule::operator()(const Cells & cells) const
{
  auto it = lookup.find(cells);
  if(it != lookup.end())
    return it->second;
  Cells return_cell = gen.generate();
  lookup[cells] = return_cell;
  return return_cell;
}
```

Crossover?

- Rule: map<Cells, Cells>

Mutation?

- Simple!
- Rule: Cells -> Cells
- Rule[random] = new random cell

Fitness?

- *whistle*
- Also
  - How do we really crossover or mutate “Just in Time” rules?
  - Left as an exercise….

Another algo

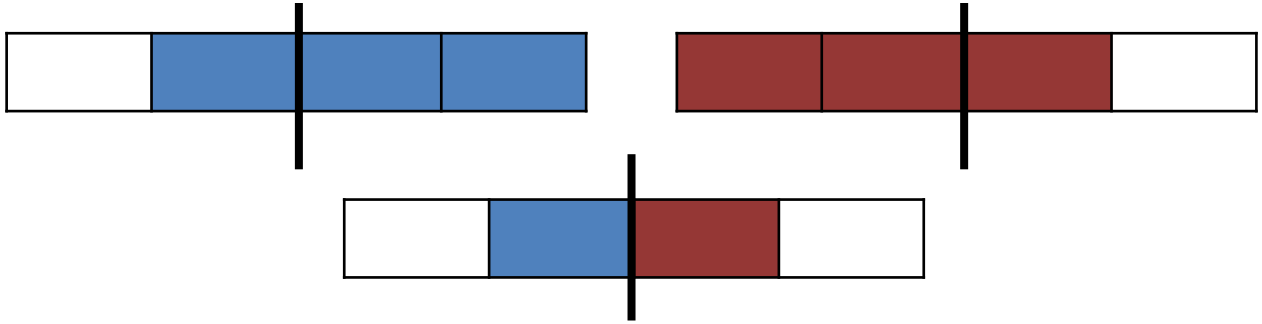
- Find the best **starting cells** for a fixed rule
- For a while
  - Generate random starting points
    - [1, 0, 0, 1]
  - Fitness = how many survive to the top
  - Report best each time
  - Crossover, mutate the starting cells

Some code

```
Rule rule = [](const Cells & cell) {
    return cell;
};

CellGenerator cell_generator(seed, cell_size);
std::vector<Cells> population;//fill using generator

for(size_t epoch = 0; epoch < 25; ++epoch) {
    for(auto cell : population)
        auto alive = run_trial(rule, cell);
    crossover(population);
    mutate(population);
}
//return best and show it
```

Crossover – who?

- Tournament
  - best of 3 (or some other magic number)
- What if the child is worse?



Fitness

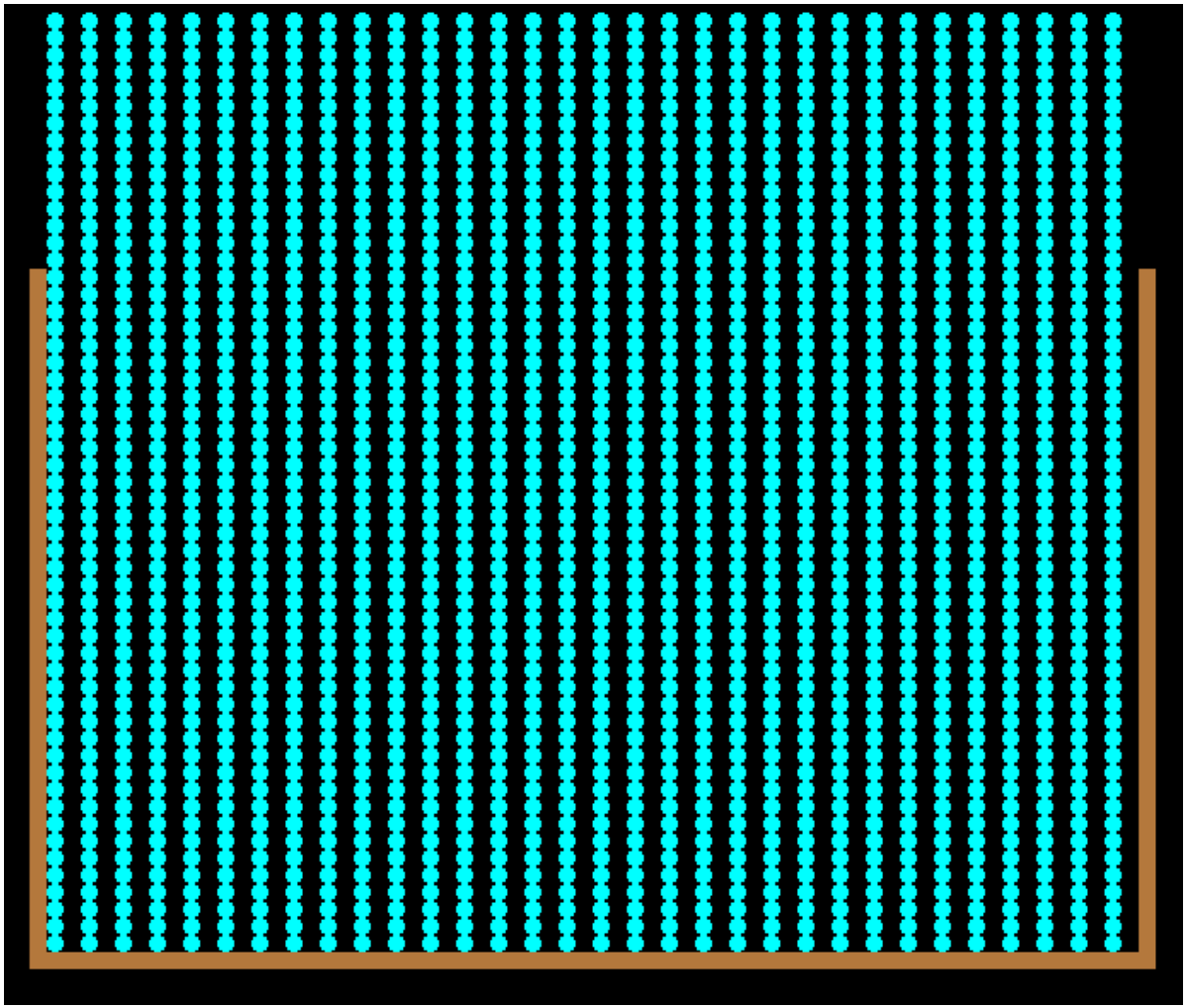
- Need to rank solutions
  - Need a number -> ordering
- Use how many alive
- Could
  - keep any that leave cells alive at top
  - keep the best one
  - vary population size
  - vary crossover point

Mutation

- Flip a bit - or several
- How often? Always? Everything?
- 50% seems good enough, via experimentation

```
Cells Mutation::mutate(Cells cell) {
    auto maybe = uniform_dist(gen);
    if (maybe < cell.size()/2) {
        auto index = uniform_dist(gen);
        cell[index] = !cell[index];
    }
    return cells;
}
```

It can find the best (sometimes)



Action…

- (Note to self – do a demo)
  - Evolution.exe gaca population=25 find_start

Questions

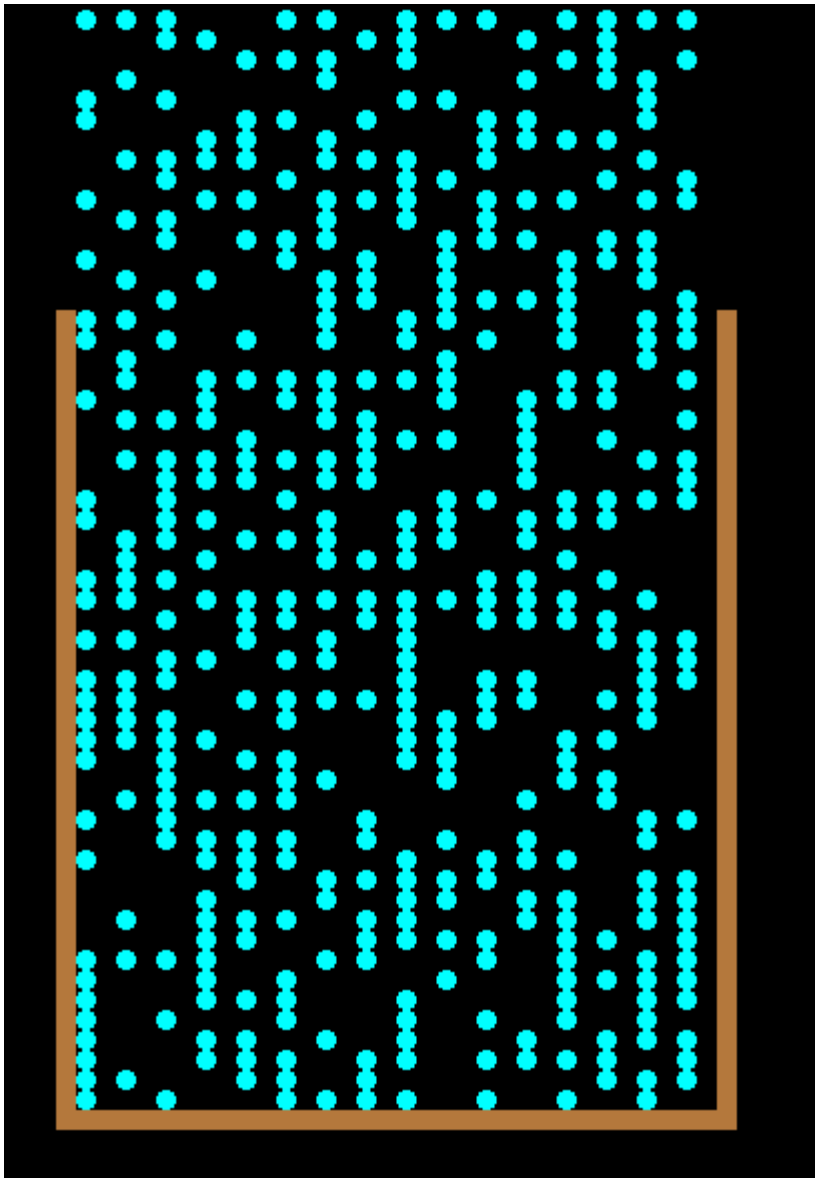
- How often does it find the best starting cell?
- What difference do the parameters make?
  - Population size
  - Epochs – and could bail early if all live
  - Mutation probability
  - Crossover point(s)
- Magic numbers!

Machines don’t learn

- We have chosen the parameters
  - epochs
  - population size
  - crossover point
  - mutation rate
- Experimentation finds the best
  - Or good enough
- We could GA these too!
  - Or similar: “metaheuristic parameter selection”

Random rule(s)

- For now we will just generate one rule
  - an exercise for the listener to define “best”
  - “human in the loop”?
- Action…
  - (Note to self - do demo)
  - Evolution.exe gaca
  - Evolution.exe gaca find_start rand_start



A world where anything is possible

Round up

- GA
- CA
- GACA
  - What's a "good" rule?
- Random trials are good
  - c.f. scratch refactoring
  - As long you can define "good"
- Computers can test (and generate code) for us