

Truly complex programs cannot be
written by humans...

Dr. Julian Miller

Intelligent Systems Group

Department of Electronics

The University of York

jfm7@ohm.york.ac.uk

<http://www.elec.york.ac.uk/staff/academic/jfm.html>

The way we design things is the problem

- How could we *design* an electronic circuit or a computer program that has 10^{13} *sophisticated* subcomponents?
 - Standard software metric is 10 GLOC per day
 - Windows 2000 has 10^8 LOC
 - To produce a program with 10^{13} GLOC would take one person 10^{12} days = 2.8 billion years
 - And a GLOC is NOT a sophisticated subcomponent! (neither is a transistor)

Someone else has noticed this problem

- "Consider this: at current rates of expansion there will not be enough skilled I/T people to keep the world's computer systems running...Some estimates for the number of I/T workers required globally...put it at 200 million, or close to the entire population of the United States. Even if we could somehow come up with enough skilled people, the complexity is growing beyond human ability to manage it...Without new approaches, things will only get worse...".

Taking a leaf from Nature's Book.

- Complexity Ceiling
 - I do **not** believe that conventional methods of design will achieve high complexity and low maintenance costs unless we embrace a biologically plausible model.
- How? We must adopt a model of design akin to the biological development of multicellular organisms. Why?
 - Developmental models can exploit emergence
 - Development is at the heart of regeneration and self-repair
 - Developmental models scale well

How can you manipulate things you don't understand?

- Humans do it all the time!
- Gardeners— plant seeds, graft and prune plants...
 - Do they know how the cells work?
- Imagine that creating a complex artifact is like growing a plant...

How can we program complex systems?

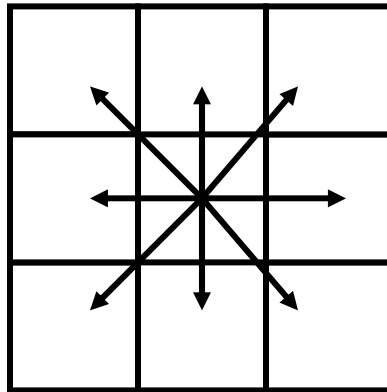
- I propose that we need a new software paradigm with the following requirements
 - A visual programming environment
 - Computational stem cells will be available from ‘seed’ catalogues
 - From these computational phenotypes will be
 - Grown, evolved, shaped, grafted
 - Desired programs will be shaped by observing their behaviour and applying variation

Visual Programming Languages: some definitions

- Programs that are specified by two or more dimensional objects
- Provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction.
- A set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world.
- Examples: Kodu, StarLogoTNG, Alice

From one cell to many cells

A single cell is placed at a point in a grid



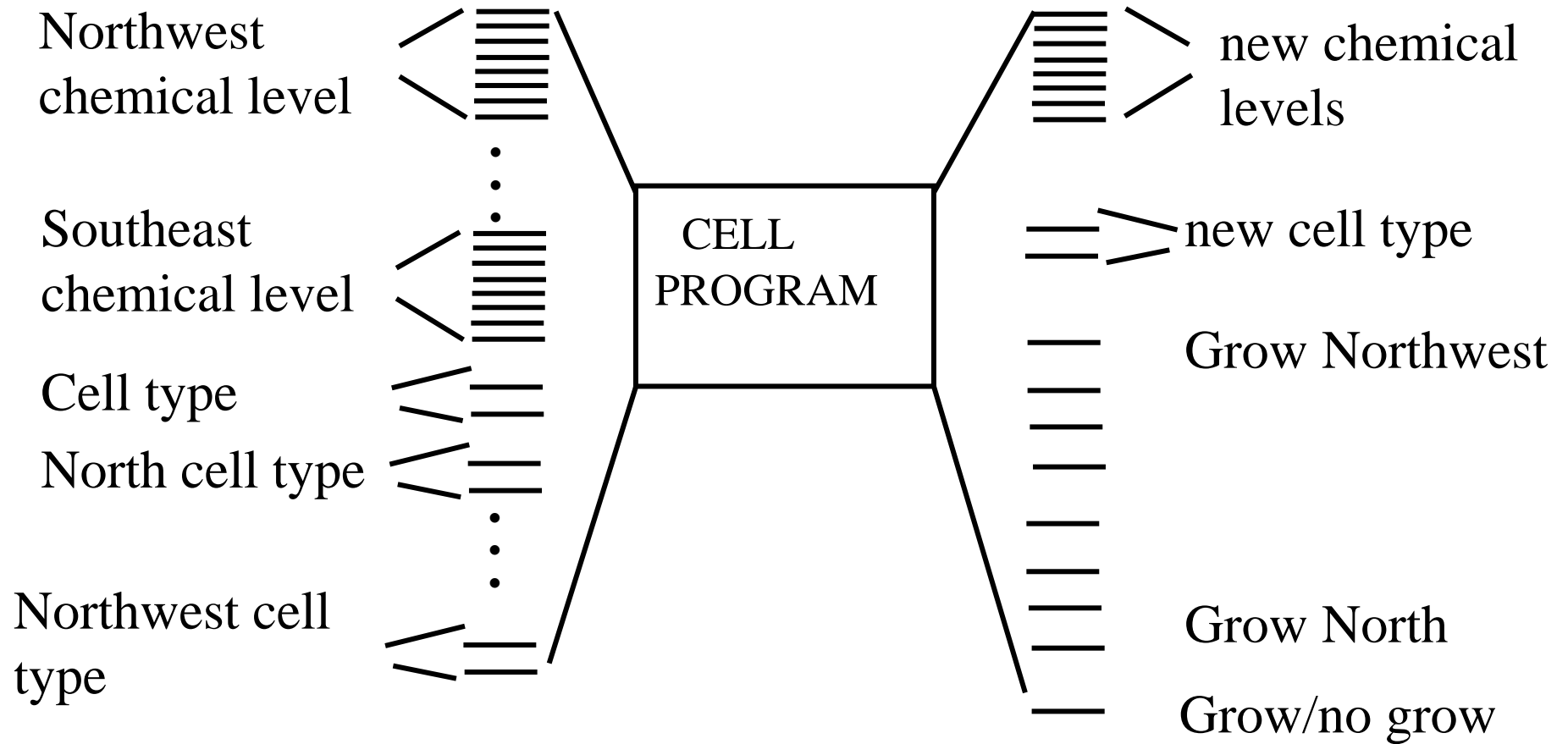
Cells can grow in 8 directions at each timestep

A cell reads its own state, the states of its neighbours and the amounts of chemical at the neighbouring states.

Cells can have a number of types, cells can die.

Chemicals obey a diffusion rule.

Program inputs and outputs

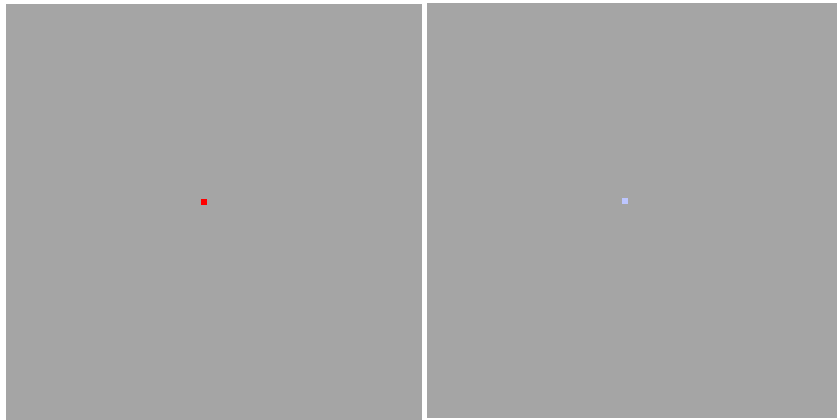


A particular case: German flag experiment

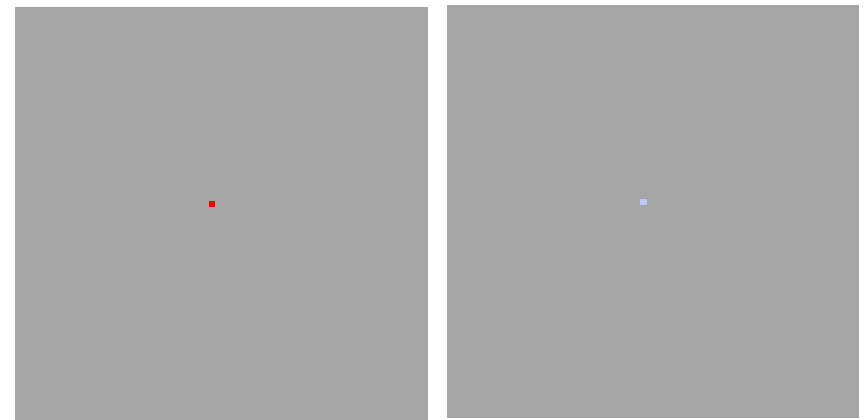
- Task: Evolve a program inside a cell that constructs a growing German flag of arbitrary size
- At iteration 4, it should look like a small flag, at iteration 6, it should look like the medium sized flag
- Fitness = sum of correct cell types at the two test points

Grafting flags

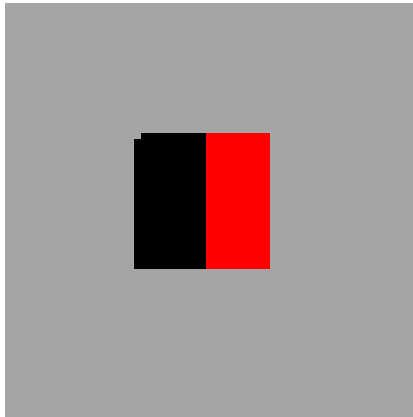
cells gGf11 chemical



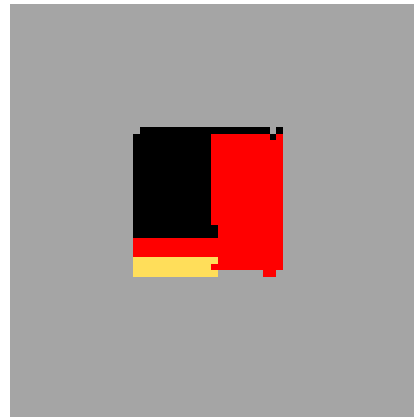
cells gGf0 chemical



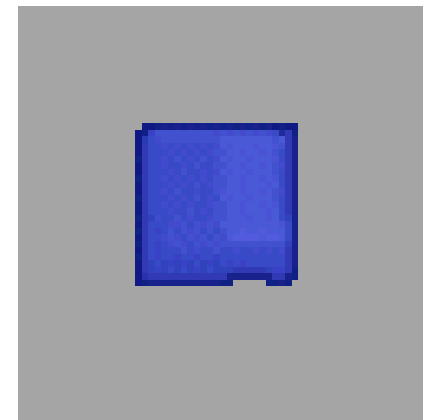
Genotype map



gGf11/gGf0 cells



chemical



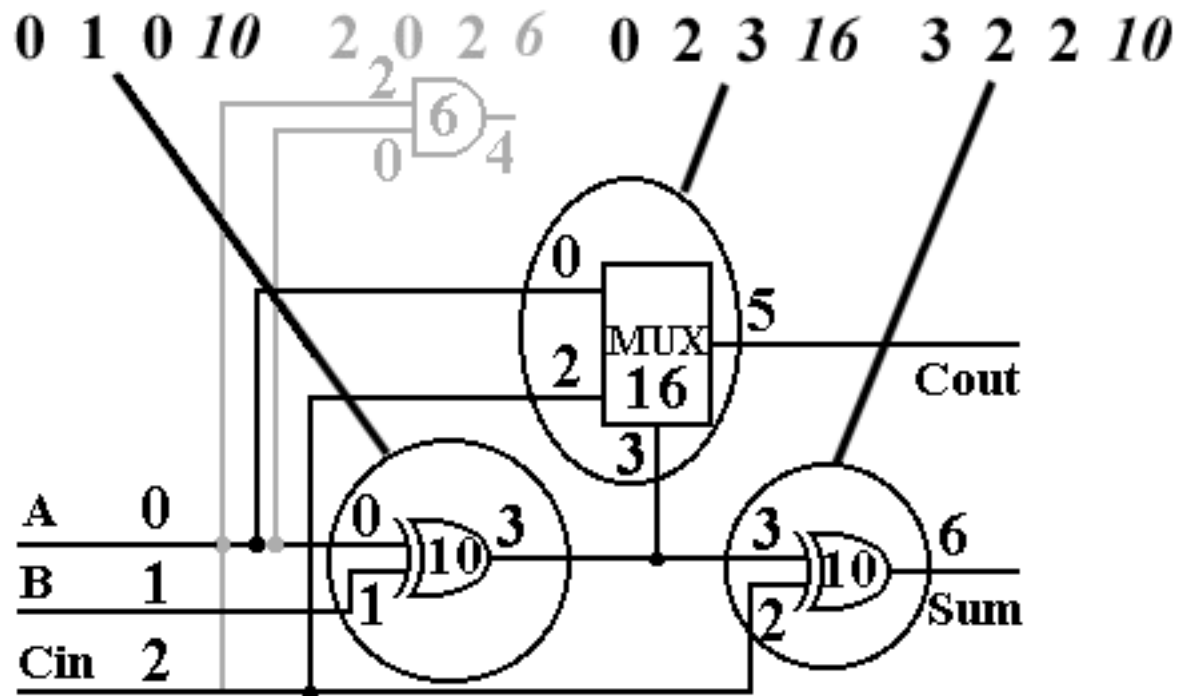
What needs to be done?

- Develop a prototype visual, growing programming language
 - Show that growing, grafting and pruning computational structures can solve problems (i.e. make it computational rather than pattern generating)
 - Show that specific computational behaviours can be evolved (e.g. robot/animat behaviour, GUI, adaptive...)
 - Evolve computational stem cell software libraries
 - Develop useful new applications

Cartesian Genetic Programming (CGP)

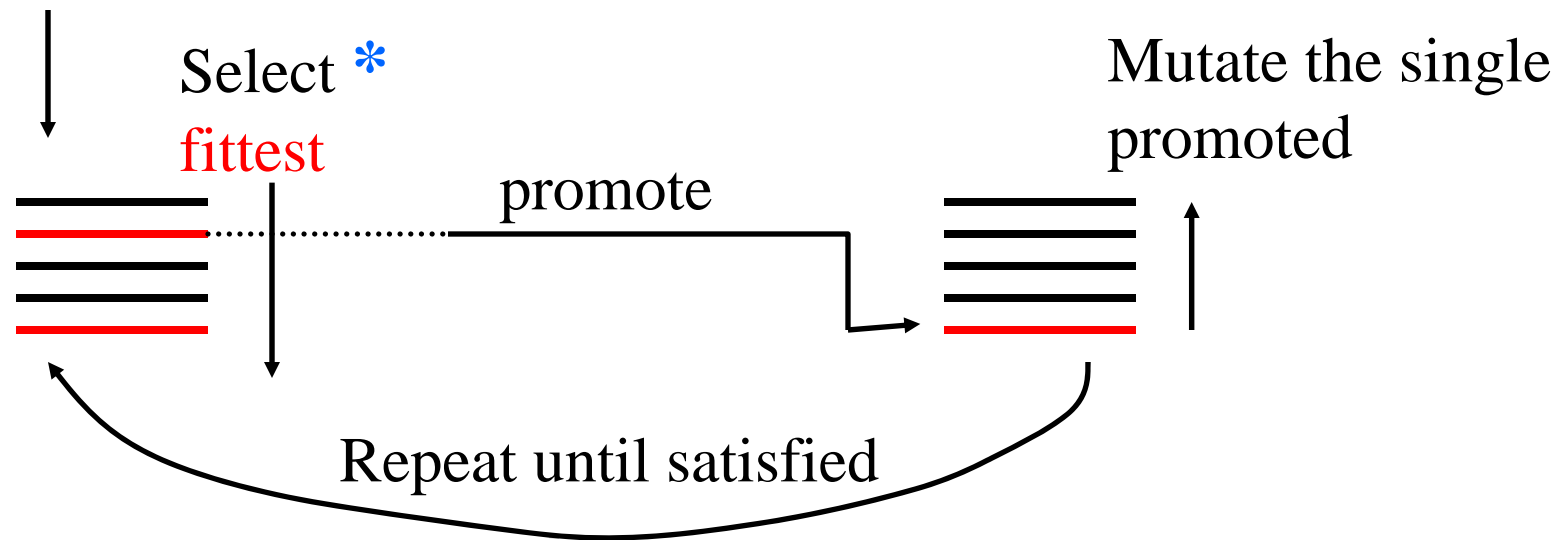
- CGP is used to encode the cell program
- Genotype is an encoding of directed graphs rather than trees
- Contains active and inactive regions (rather than coding or non-coding)
- Mutations can make active genes become inactive and inactive genes become active

Example: 1-bit adder with carry



A simple evolutionary algorithm

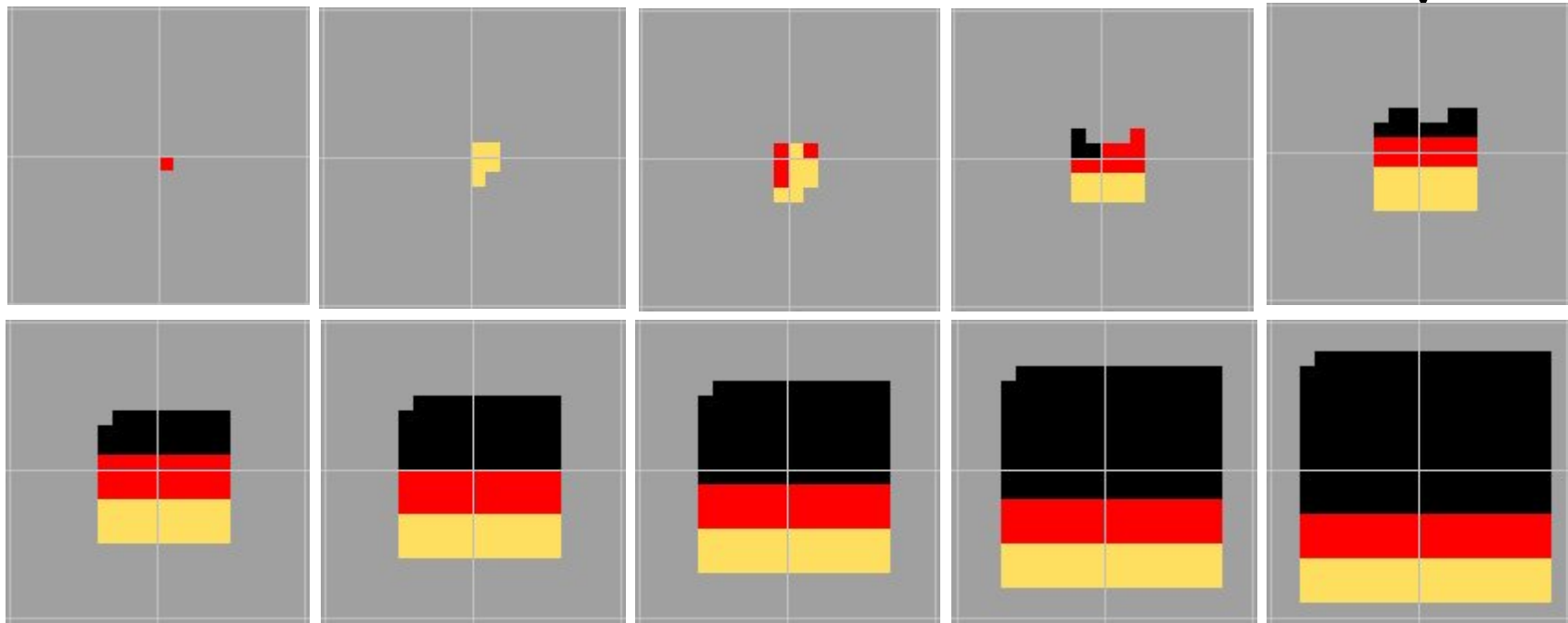
Population of
genotypes (5)



* if no unique fittest, then promote an equally fit genotype

An interesting solution (gGf11)

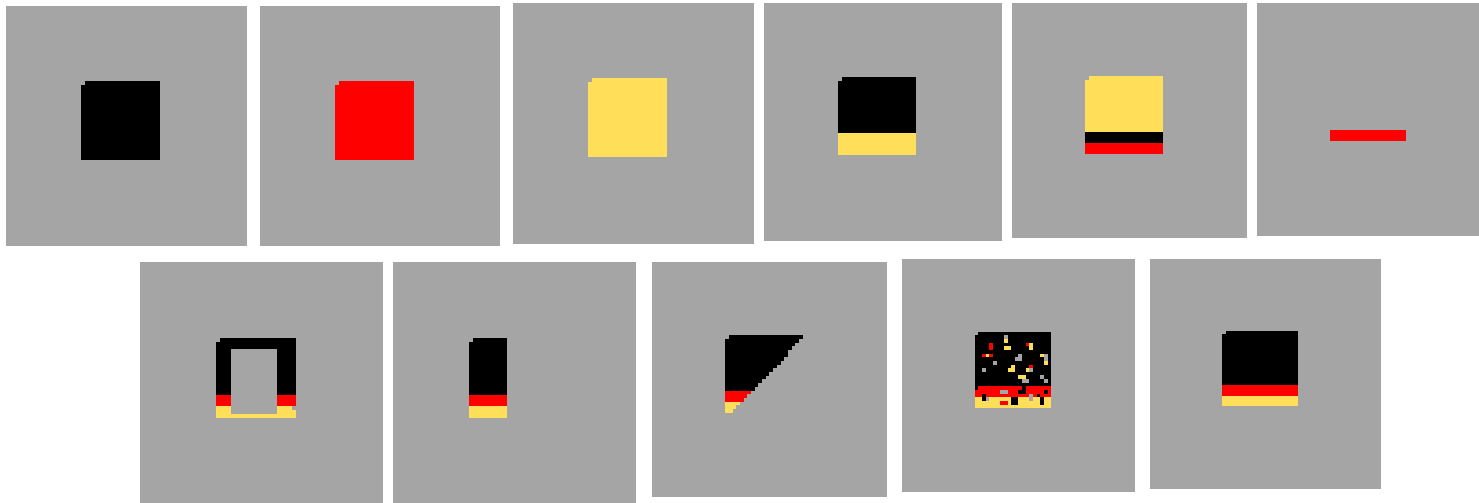
Iteration 4



Iteration 6

Regeneration experiments on gGf11

- The German flag organism is altered or damaged at iteration 11 and then allowed to grow (re-running original cell program). The organism shows a remarkable stability and sometimes an emergent ability to self-repair and regenerate

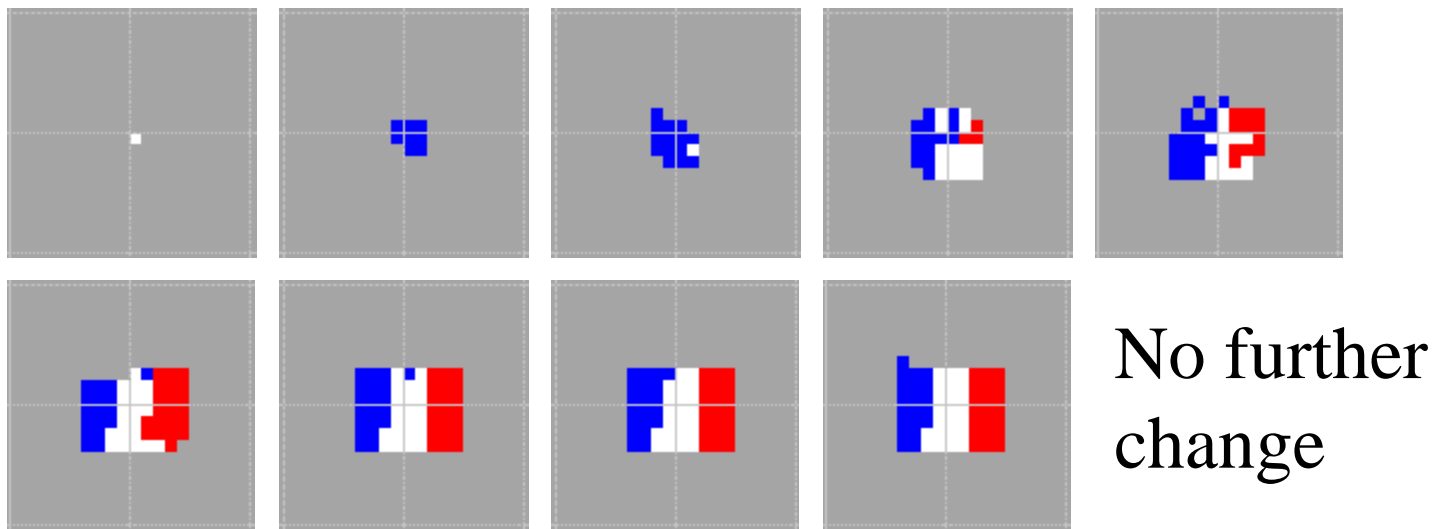


French flag experiment

- Task: Evolve a program inside a cell that constructs a French flag of fixed size (i.e. it grows and then stops growing)
- At iterations 5-8 it should look like the French flag
- Fitness = sum of correct cell types at the four test points

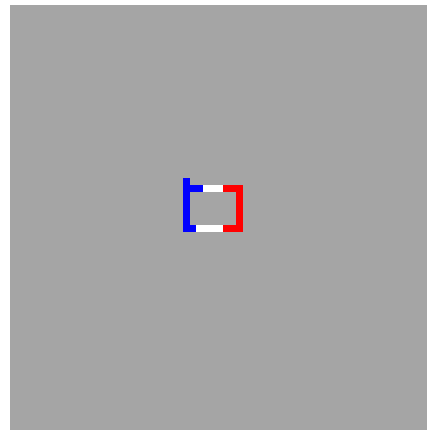
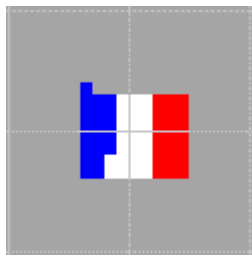
Growing, Maturing, Regenerating French Flag

- Here is an evolved program developing into a French flag like multicellular organism

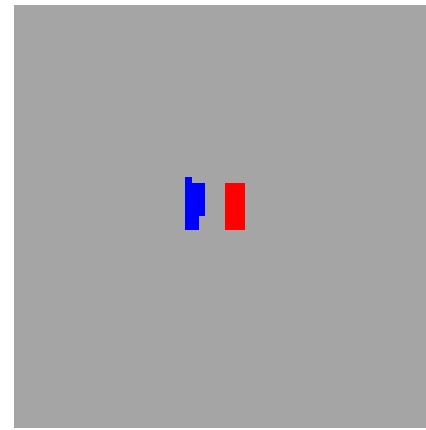


Behaviours of maturing French Flag when damaged

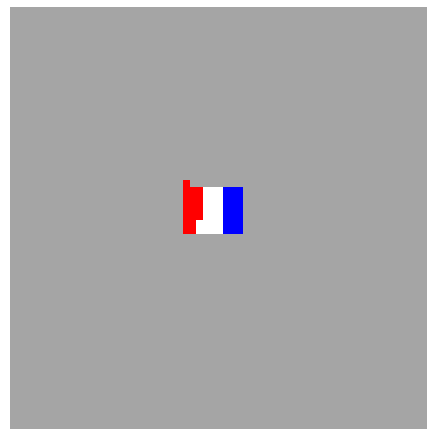
Original
mature Ff



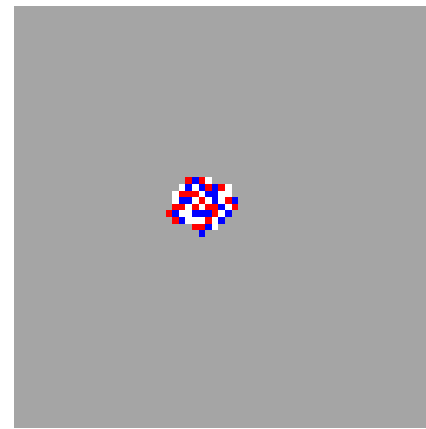
Big hole



White removed



Red and blue swapped



Randomised

Example: Kodu

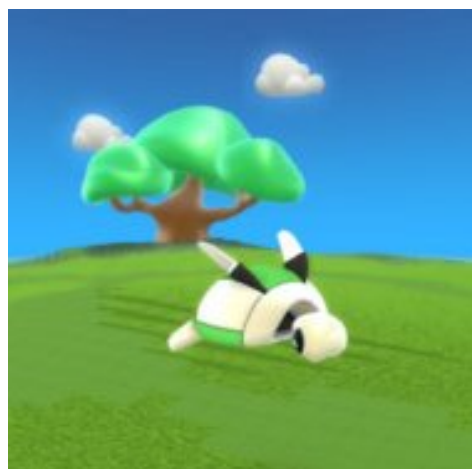
- The core of the Kodu project is the programming user interface. The Kodu programming language is simple and entirely icon-based.
 - Programs are composed of pages, which are broken down into rules, which are further divided into conditions and actions. Conditions are evaluated simultaneously.
- Programs are expressed in physical terms, using concepts like vision, hearing, and time to control the behavior of an animated character.

Source: <http://research.microsoft.com/en-us/projects/kodu/>

Kodu Screenshots



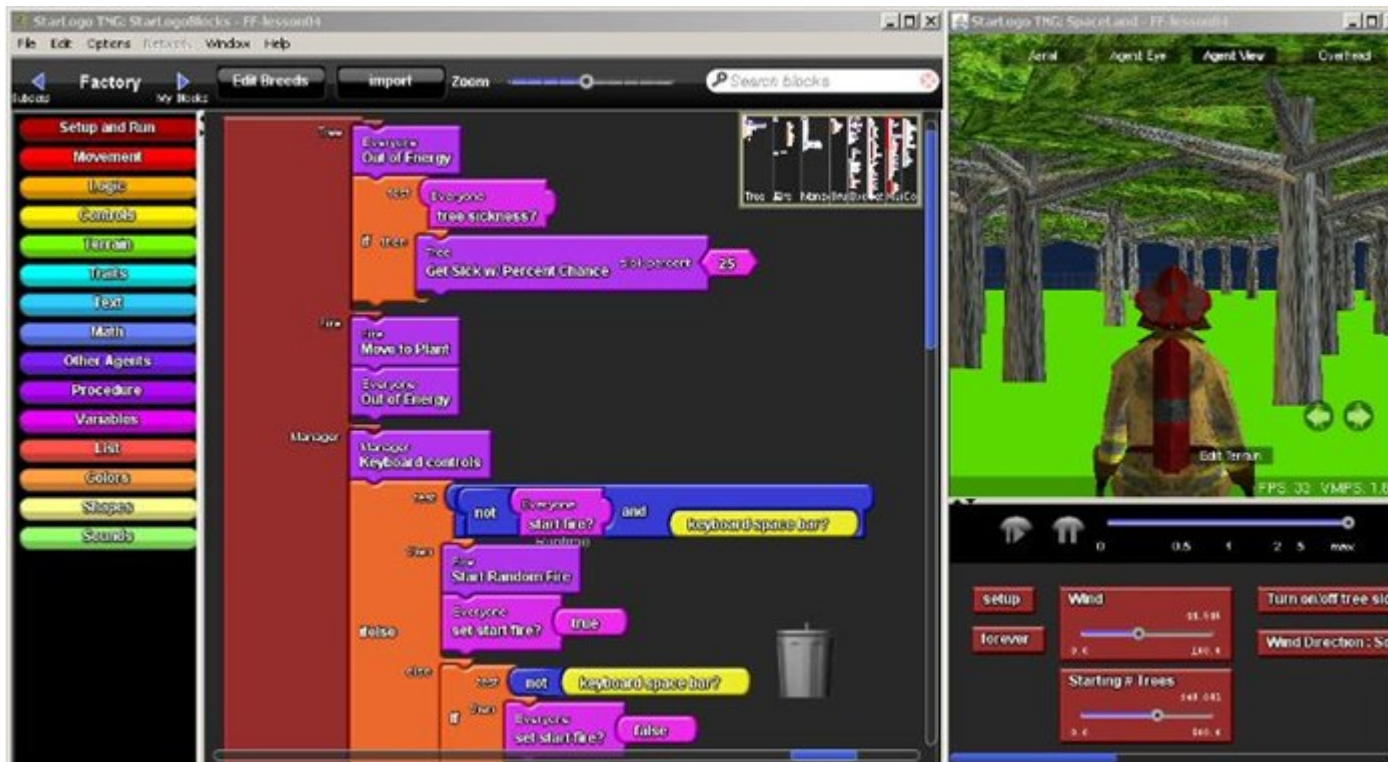
"Physical" sensors are used as rule input.



Example: StarLogo TNG

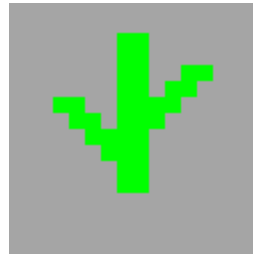
- StarLogo TNG is The Next Generation of StarLogo modeling and simulation software. It is a tool to create and understand simulations of complex systems, it also brings with it several advances.
 - Lower the barrier to entry for programming with a graphical interface where language elements are represented by coloured blocks that fit together like pieces of a puzzle.

StarLogo TNG screenshot



Maturing plant experiment

- Try to evolve a program inside a cell that grows into a 32 cell organism that looks like a branch
- Fitness method:
 - Compare the organism cell by cell with a predefined “perfect” plant at iterations 7, 8, 9



Defined on 16x16
grid of cells