TechTalk on Artificial Intelligence

— A practical approach to Genetic Algorithm —

Alexandre Bergel
University of Chile, Object Profile
http://bergel.eu

Goal of today

Give an introduction to what genetic algorithm is

Show what can be done in plain Pharo related to genetic algorithm

These slides…

… are a support for the TechTalk

… are not meant to be understandable when read offline

… are a summary of a lecture given at the University of Chile

… incremental in their content

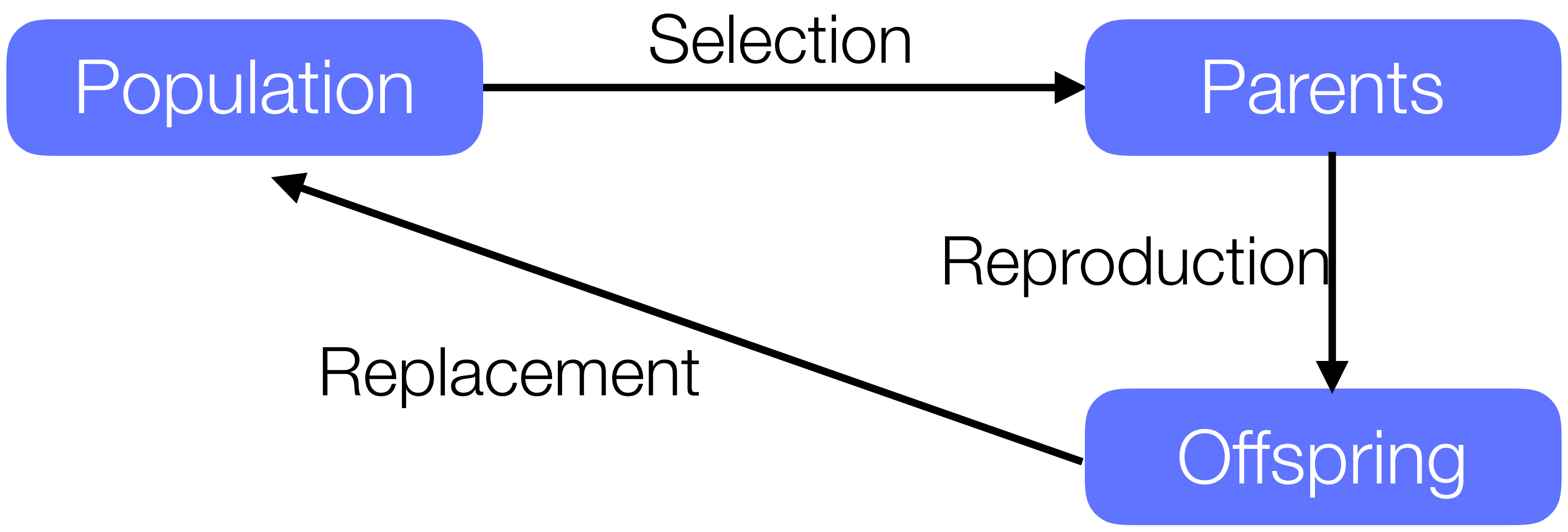
“The fact that life evolved out of nearly nothing, some 10 billion years after the universe evolved out of literally nothing, is a fact so staggering that I would be mad to attempt words to do it justice.”

— Richard Dawkins

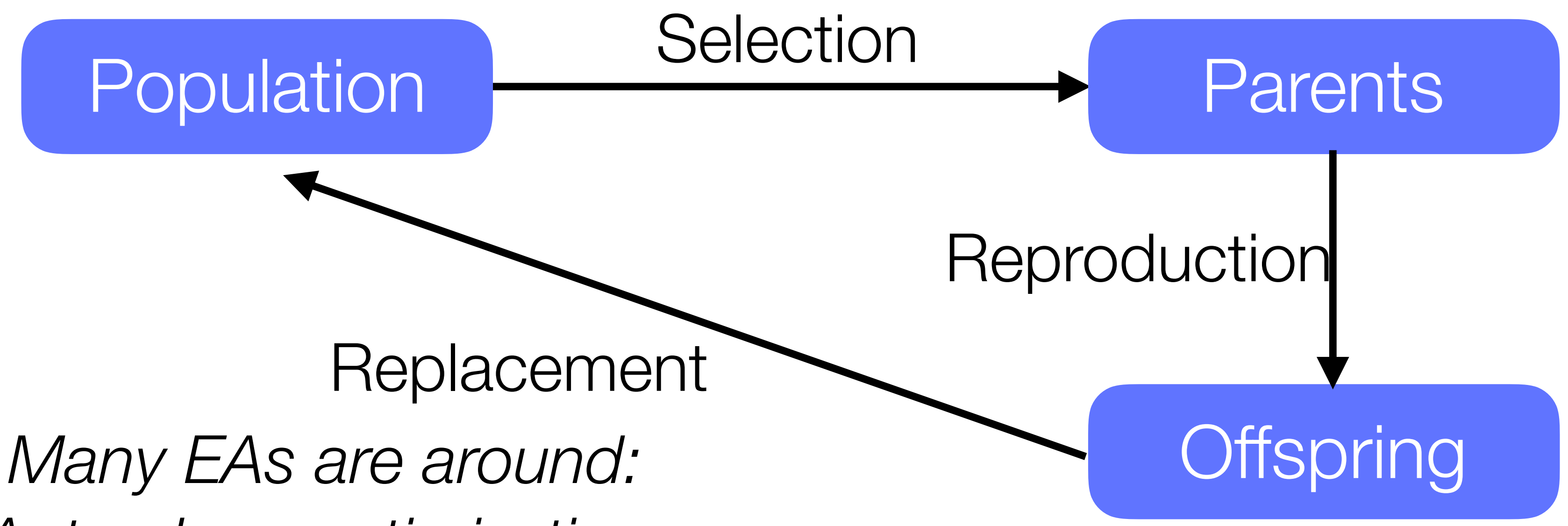
“One general law, leading to the advancement of all organic beings, namely, vary, let the strongest live and the weakest die”

— Charles Darwin

Flow - chart of an evolution algorithm



Flow - chart of an evolution algorithm



Many EAs are around:
Ant colony optimization,
Artificial immune system,
Cultural algorithms,
Genetic Algorithm,
Genetic Programming,
Grammatical evolution,
…

Evolution algorithm

May be written as:

x[t + 1] = v(s(x[t]))

where:

x[t] is the population at time t under a representation x

s is the selection operator

v is a random variation operator

Evolution algorithm

Evolutionary algorithms have many advantages, including:

- Offer a framework such that it is comparably easy to incorporate prior knowledge about the problem.
- May be combined with other optimization techniques. For example, could be used to tuning weights in a neural networks

Genetic Algorithm in a Nutshell

Evolutionary computation technique that automatically solves problems without specifying the form or structure of the solution in advance

Generally speaking, *genetic algorithms are simulations of evolution*, using biological genetic operations

Genetic Algorithm in a Nutshell

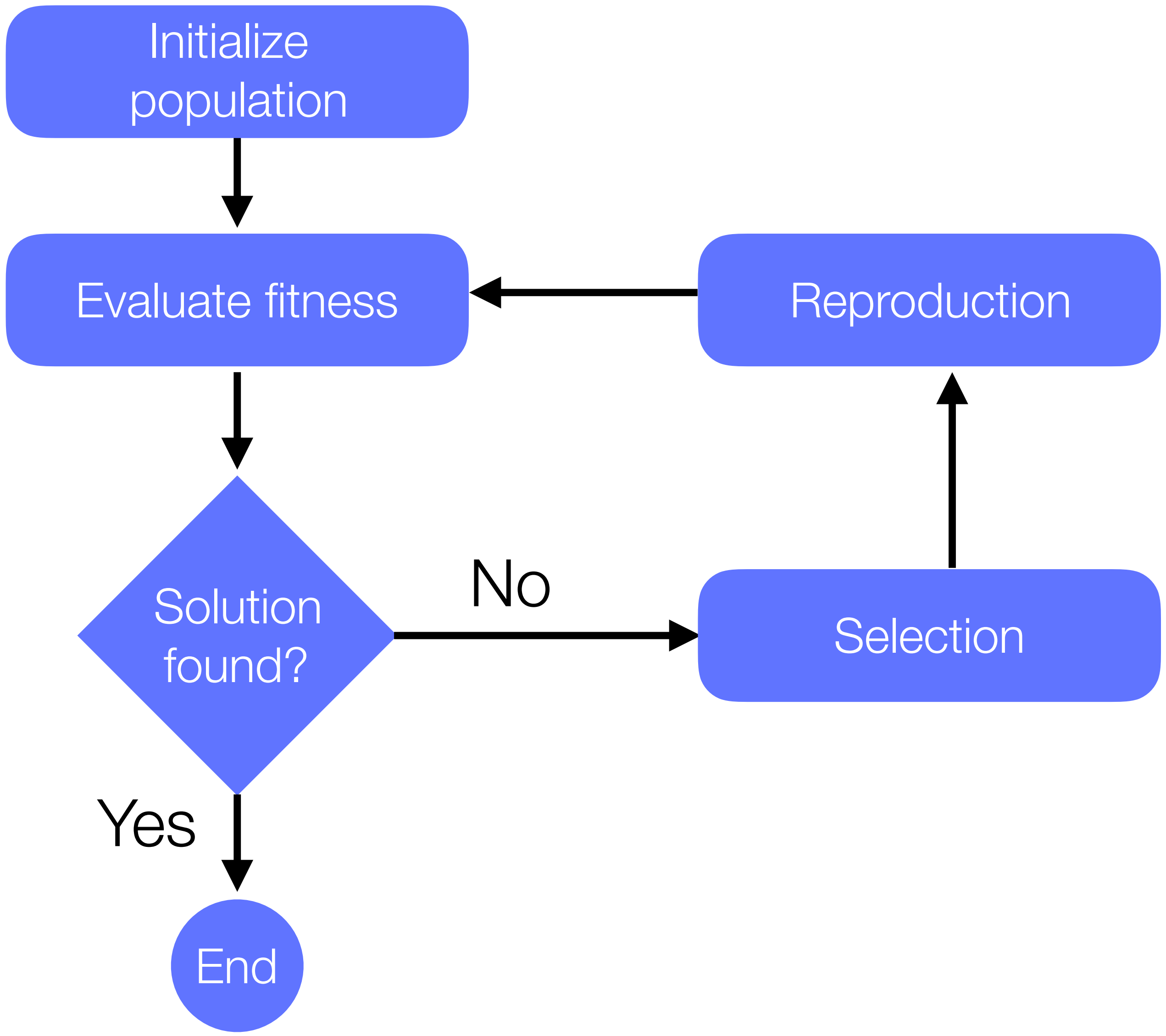
The idea first appears in 1967, in J. D. Bagley's thesis "*The Behavior of Adaptive Systems Which Employ Genetic and Correlative Algorithms*"

Since then, this field has witnessed a tremendous development

Often considered as an optimization method

- ie. finding x such as f : X -> R is maximal, x belonging to X, a multidimensional space

Flow - chart of a genetic algorithm



The algorithm

```
Compute initial population $\mathcal{B}_0$;

WHILE stopping condition not fulfilled DO
BEGIN
    select individuals for reproduction;
    create offsprings by crossing individuals;
    eventually mutate some individuals;
    compute new generation
END
```

Canonical example

A friend asks you to solve a challenge:

- He secretly wrote a word of 3 letters, and challenge you to find it

Your friend can help you that way:

- He can tell you how many letters are actually correct.

Canonical example

You have made 3 words: w1, w2, w3

Your friend tells you that 3 letters are different in w1, 2 in w2, and 1 in w3.

The sequences w2 and w3 are closer to the solution than w1

You can forget w1 as it is too far from your friend secret

You have several options:

- You can randomly create a new sequence
- You can randomly modify some letters in w2 or w3
- You can combine w2 and w3 in the hope to produce a better sequence

Step 1 - Initialize Population

This step is rather easy. It simply consists in creating N words of 3 letters

N is a parameter of your algorithm.

Let’s say N = 10

We call a sequence that belongs to our population as *individual* or *member*

Step 2 - Evaluate fitness

The fitness function produce a number score to describe the fitness of a given member of the population

GA is used to evolve a population to an optimal solution to a problem, so we need to numerically evaluate any given possible solution

Step 2 - Evaluate fitness

For example, if the secret sequence is: “cat”

We have:

f(“cow”) = 1

f(“cak”) = 2

f(“cat”) = 3

Assuming that none of our 10 sequences exactly match the secret sequence, we need to create a new generation of sequences

We therefore have to enter the selection process

Darwinian Natural Selection

In order to have a natural selection, we need to have:

Heredity: a child receives properties of its parents. In particular, if the parents are robust and can live long enough, then the child should too

Variation: some variation may be introduced in children. Children should not be identical copy of their parents

Selection: some members of a population must have the opportunity to be parents and have offsprings in order to pass their genetic information. Typically referred to as “survival of the fittest”

Step 3 - Selection

Once the fitness is computed for each individual (i.e., sequence of our population) we need to select which individuals are fit enough to become parent

Several strategies are possible:

- Pick the fittest 25%
- The probability to pick a parent depends on its fitness (e.g., if i1 has a fitness of 5 and i2 a fitness of 10, then i2 has 2 times more probability to be picked)

This step result in a *mating pool*, in which parent will be picked in the following step

Step 3 - Selection

A generic selection procedure may be implemented as follows:

1. The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1.
2. The population is sorted by descending fitness values.
3. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should be 1 (otherwise something went wrong in the normalization step).
4. A random number *R* between 0 and 1 is chosen.
5. The selected individual is the last one whose accumulated normalized value is smaller than *R*.

Step 4 - Reproduction

This step builds a new population (with the same size)

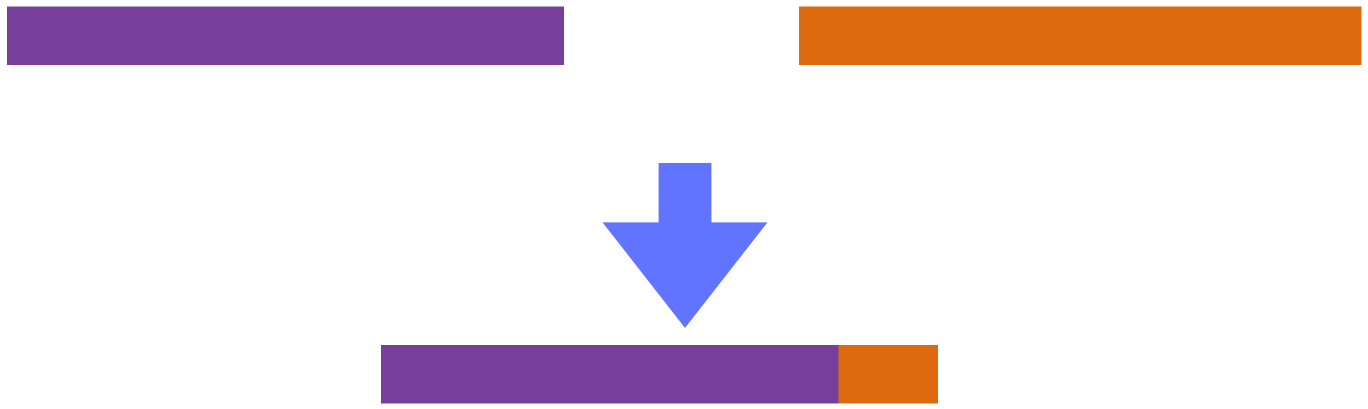
Individuals composing this new populations are “babies” of two parents individuals (from the previous population)

Each baby is created using two genetic operations: cross-over and mutation

Step 4 - Reproduction

Pick two parents from the mating pool

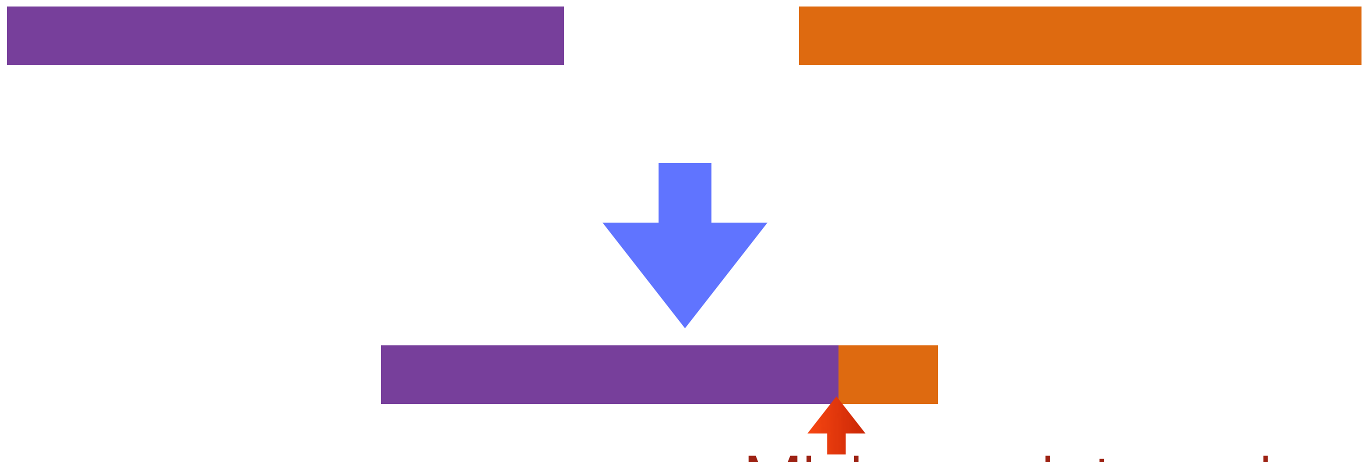
Create a new individual, for which its genes is the result of mixing the parents' genes



Step 4 - Reproduction

Pick two parents from the mating pool

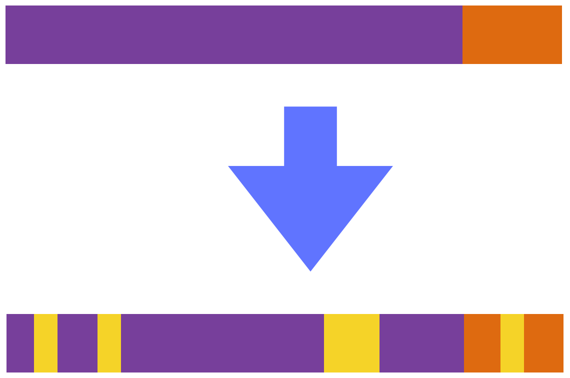
Create a new individual, for which its genes is the result of mixing the parents' genes



Mixing point randomly picked

Step 4 - Reproduction

Go over each gene of the child if replace the gene using a mutationRate probability



Configuring the algorithm

Mutation rate: % to change a gene when creating a child

Population size: number of individual to consider each time

Number of genes: how many genes contains each individual

Fitness function: Function that tells how good / far an individual is from the (ideal) solution

Result of the algorithm

Once the algorithm is run, we need to know how good we did

Two metrics are usually enough:

- Number of generations until the solution is found
- Total time until the solution is found

The Fitness Function

The real hard work of doing some genetic algorithm is to write the fitness function

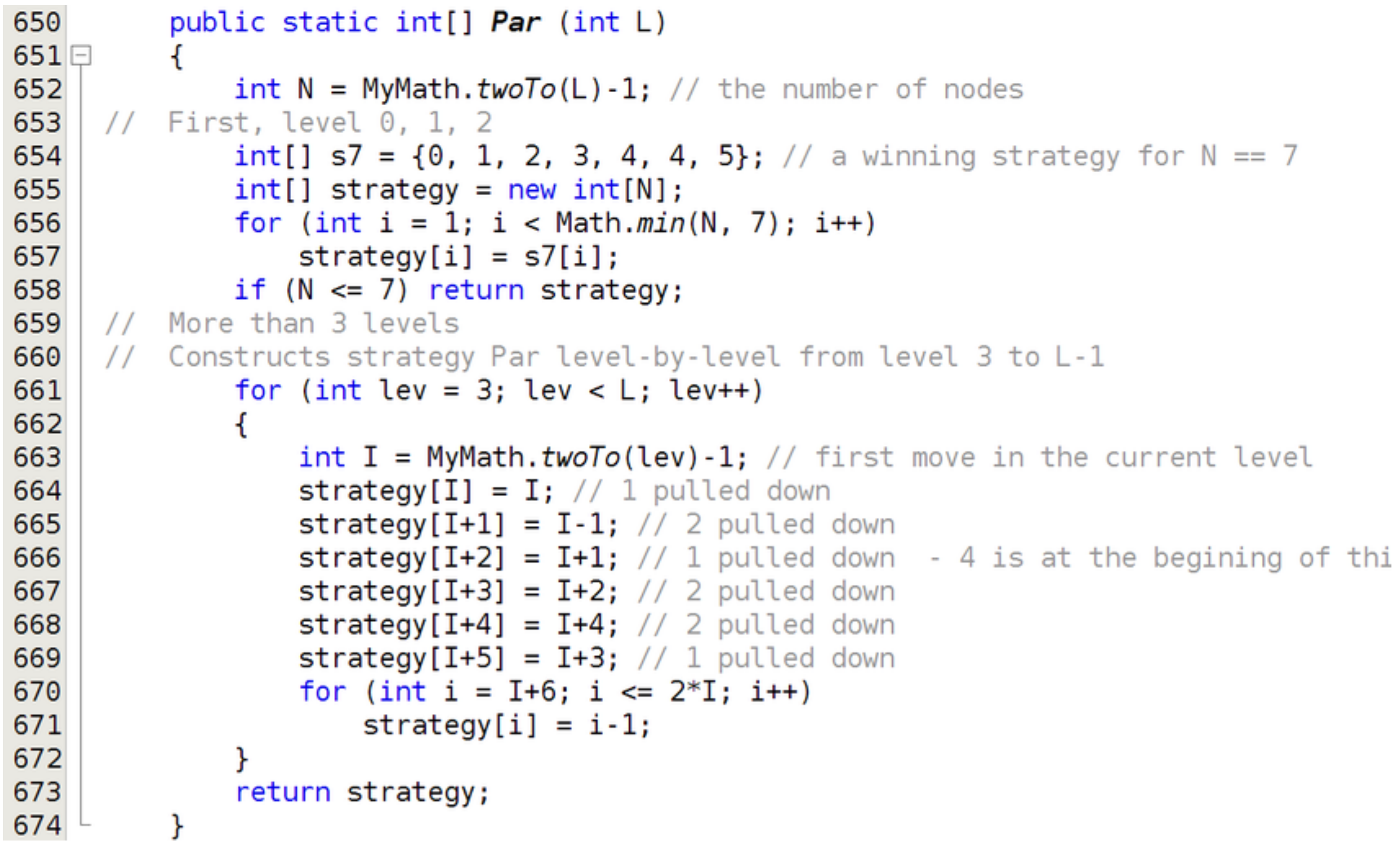
The function describes the goal and how well an individual

Demo

```
Gofer it
    smalltalkhubUser: 'abergel' project: 'GeneticAlgo';
    configurationOf: 'GeneticAlgo';
    loadDevelopment
```

Applications:
Software Performance

What is the performance of this car?



```
    public static int[] Par (int L)
    {
        int N = MyMath.twoTo(L)-1; // the number of nodes
//  First, level 0, 1, 2
        int[] s7 = {0, 1, 2, 3, 4, 4, 5}; // a winning strategy for N == 7
        int[] strategy = new int[N];
        for (int i = 1; i < Math.min(N, 7); i++)
            strategy[i] = s7[i];
        if (N <= 7) return strategy;
//  More than 3 levels
//  Constructs strategy Par level-by-level from level 3 to L-1
        for (int lev = 3; lev < L; lev++)
        {
            int I = MyMath.twoTo(lev)-1; // first move in the current level
            strategy[I] = I; // 1 pulled down
            strategy[I+1] = I-1; // 2 pulled down
            strategy[I+2] = I+1; // 1 pulled down  - 4 is at the begining of thi
            strategy[I+3] = I+2; // 2 pulled down
            strategy[I+4] = I+4; // 2 pulled down
            strategy[I+5] = I+3; // 1 pulled down
            for (int i = I+6; i <= 2*I; i++)
                strategy[i] = i-1;
        }
        return strategy;
    }
```

What is the performance of this source code?

Problem description

A *benchmark* is a representative execution of a software system

Essential to measure performance evolution

A benchmark requires a *workload*

However, defining such workload is not trivial

```java
public class CSVImporter {
    private ArrayList<ArrayList<Double>> content
                                   = new ArrayList<>();

    private void importFrom(BufferedReader r)
    throws IOException {
        String row;
        while((row = r.readLine()) != null) {
            ArrayList<Double> fs = new ArrayList<Double>();
            for(String value : row.split(","))
                fs.add(Double.parseDouble(value));
            content.add(fs);
        }
    }

    public void importFrom(String filename)
    throws IOException {
        FileReader fr = new FileReader(filename);
        this.importFrom(new BufferedReader(fr));
    }
}
```

```
7.398087255376432,3.076587022783508,3.088394050993209
9.982048943630442,4.777197718982212,0.23295837931007068
0.07317507642934801,0.8468390353242117,0.9063179655495648
5.257870939214654,2.126741512737582,1.9651785408915852
```

Each row line is parsed

```
public class CSVImporter {
        private ArrayList<ArrayList<Double>> content
                                        = new ArrayList<>();

        private void importFrom(BufferedReader r)
        throws IOException {
                String row;
                while((row = r.readLine()) != null) {
                        ArrayList<Double> fs = new ArrayList<Double>();
                        for(String value : row.split(","))
                                fs.add(Double.parseDouble(value));
                        content.add(fs);
                }
        }

        public void importFrom(String filename)
        throws IOException {
                FileReader fr = new FileReader(filename);
                this.importFrom(new BufferedReader(fr));
        }
}
```

```
7.398087255376432,3.076587022783508,3.088394050993209
9.982048943630442,4.777197718982212,0.23295837931007068
0.07317507642934801,0.8468390353242117,0.9063179655495648
5.257870939214654,2.126741512737582,1.9651785408915852
```

Measuring the performance of this class requires a workload

```
public class CSVImporter {
	private ArrayList<ArrayList<Double>> content
	                              = new ArrayList<>();

	private void importFrom(BufferedReader r)
	throws IOException {
		String row;
		while((row = r.readLine()) != null) {
			ArrayList<Double> fs = new ArrayList<Double>();
			for(String value : row.split(","))
				fs.add(Double.parseDouble(value));
			content.add(fs);
		}
	}

	public void importFrom(String filename)
	throws IOException {
		FileReader fr = new FileReader(filename);
		this.importFrom(new BufferedReader(fr));
	}
}
```

```
public class CSVImporter {
    private ArrayList<ArrayList<Double>> content
                    = new ArrayList<>();

    private void importFrom(BufferedReader r)
    throws IOException {
        String row;
        while((row = r.readLine()) != null) {
            ArrayList<Double> fs = new ArrayList<Double>();
            for(String value : row.split(","))
                fs.add(Double.parseDouble(value));
            content.add(fs);
        }
    }

    public void importFrom(String filename)
    throws IOException {
        FileReader fr = new FileReader(filename);
        this.importFrom(new BufferedReader(fr));
    }
}
```

```
//input1.csv

1.1,3.0

3.2,2.0
```

```
//input2.csv

1.123232,3.000001

3.21231,2.0000001
```

```
public class CSVImporter {
    private ArrayList<ArrayList<Double>> content
                    = new ArrayList<>();

    private void importFrom(BufferedReader r)
    throws IOException {
        String row;
        while((row = r.readLine()) != null) {
            ArrayList<Double> fs = new ArrayList<Double>();
            for(String value : row.split(","))
                fs.add(Double.parseDouble(value));
            content.add(fs);
        }
    }

    public void importFrom(String filename)
    throws IOException {
        FileReader fr = new FileReader(filename);
        this.importFrom(new BufferedReader(fr));
    }
}
```

```
//input1.csv

1.1,3.0

3.2,2.0
```

These two files do not take the same time to be parsed

```
//input2.csv

1.123232,3.000001

3.21231,2.0000001
```

```
//input1.csv
1.1,3.0
3.2,2.0
```

Around 5 millions numbers per second

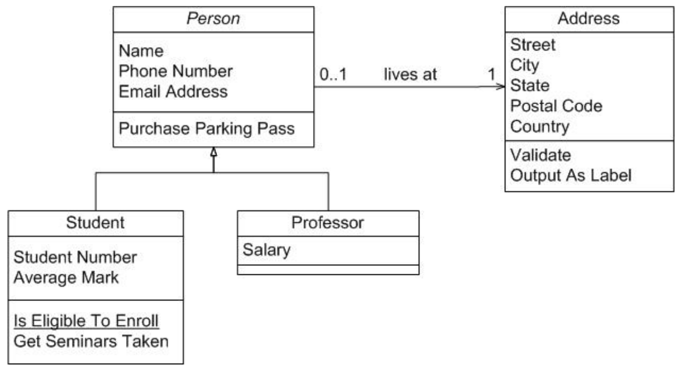
```
//input2.csv
1.123232,3.000001
3.21231,2.0000001
```

Around 3 millions numbers per second

Case study

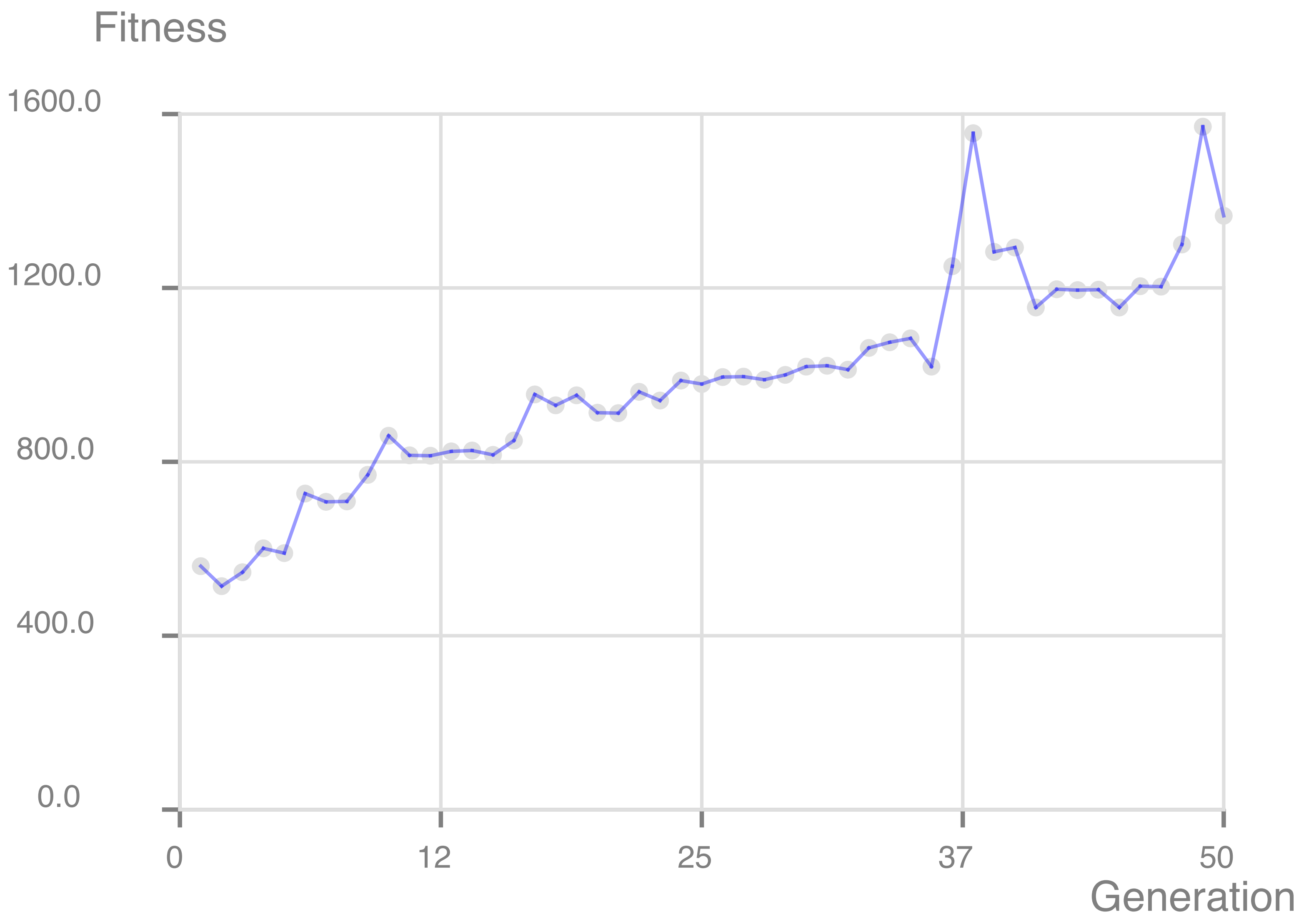
RTUMLClassBuilder is a tool to draw UML class diagrams

`SourceCode.java`

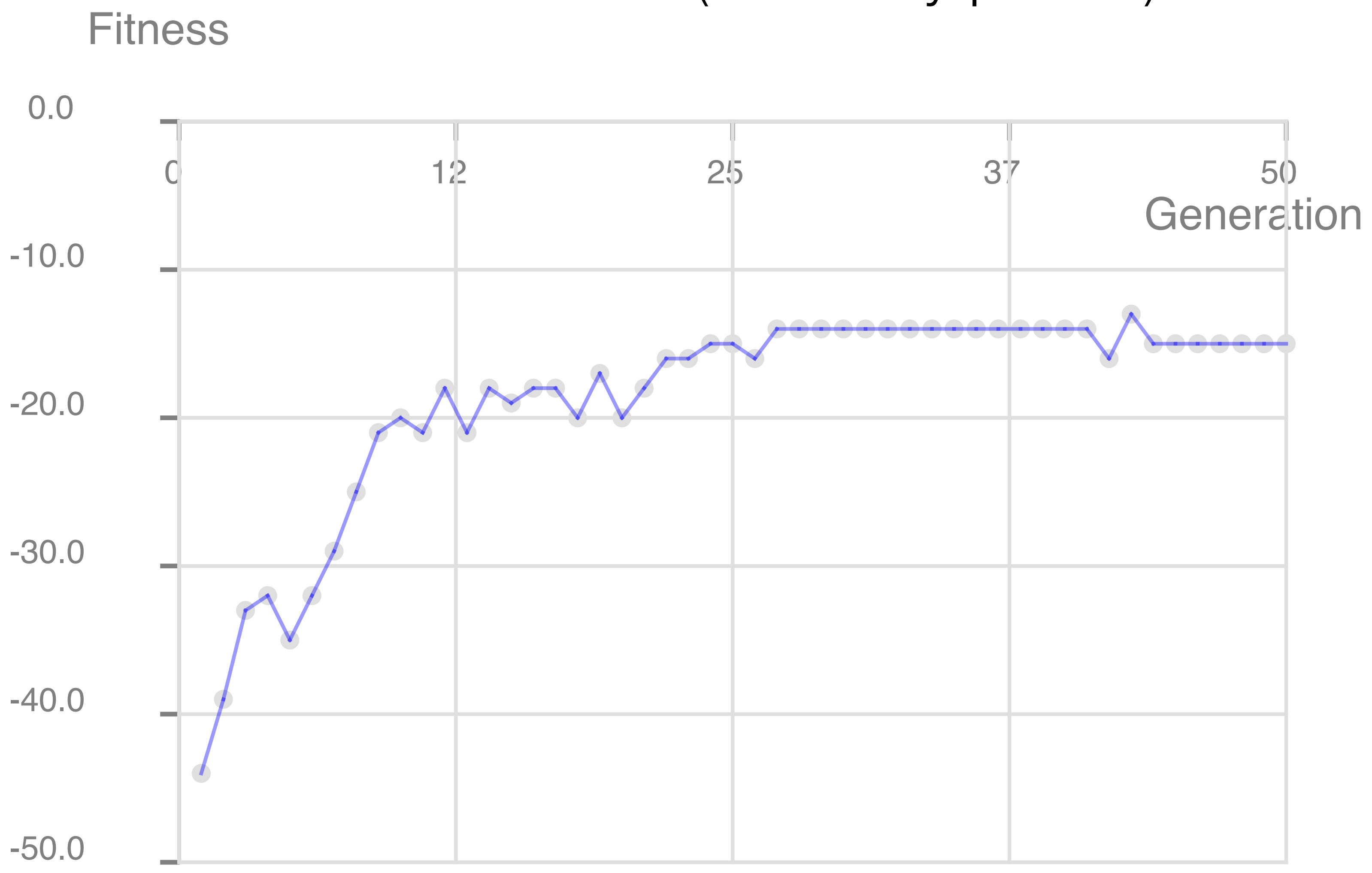


How many classes per seconds can it render?

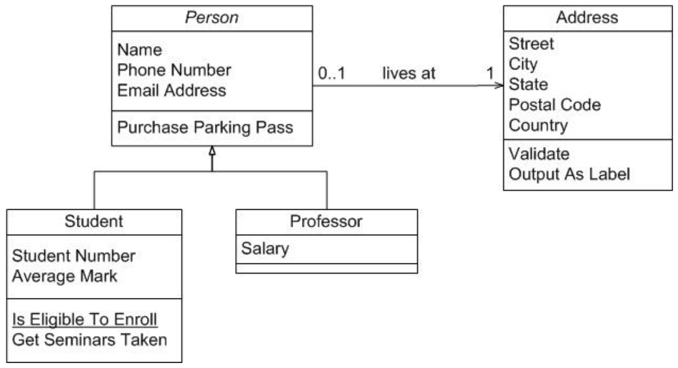
Finding the upper bound
Fitness function = time to render 100 genes
Gene = 1 class (randomly picked)



Finding the lower bound
Fitness function = negated time to render 100 genes
Gene = 1 class (randomly picked)



RTUMLClassBuilder



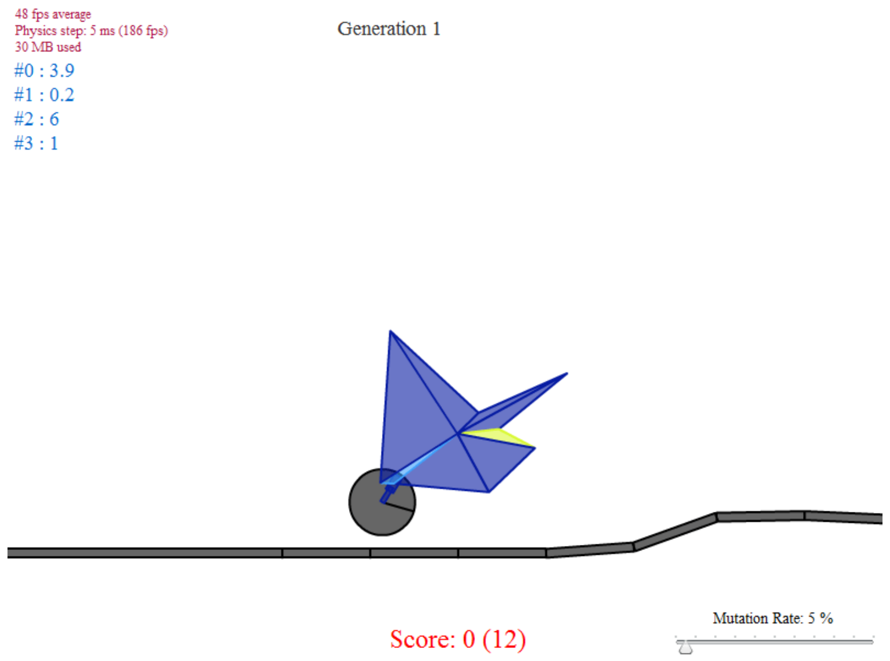
We identified the spectrum performance for RTUMLClassBuilder
In particular, we found:

- 100 classes that maximize the performance
- 100 other classes that minimize the performance

Applications:
Organic living systems
Unsupervised learning

Making a random car drive a road

http://boxcar2d.com/about.html



48 fps average
Physics step: 5 ms (186 fps)
30 MB used
#0 : 3.9
#1 : 0.2
#2 : 6
#3 : 1

Generation 1

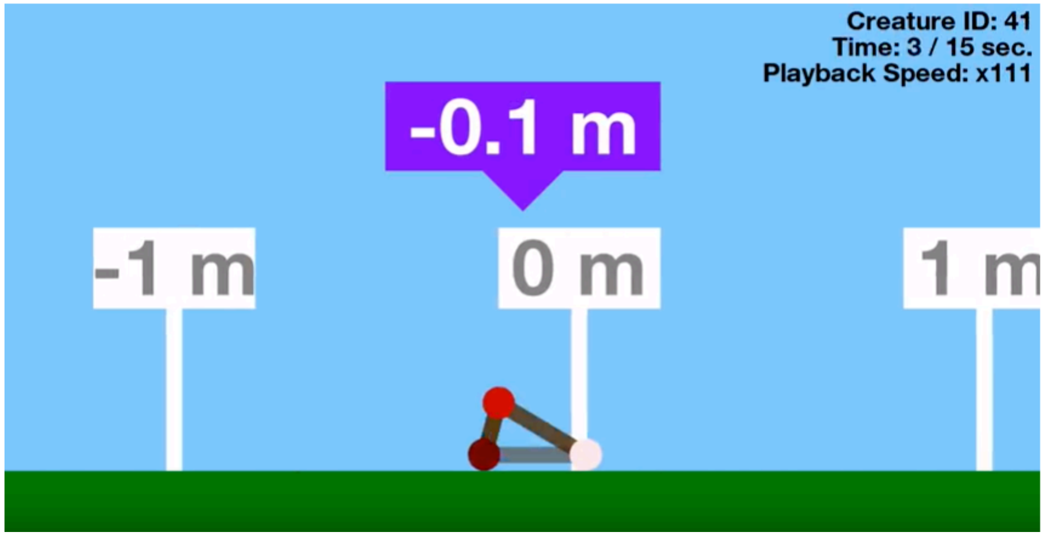
Score: 0 (12)

Mutation Rate: 5 %

Join segmented Line creatures

https://www.youtube.com/watch?v=GOFws_hhZs8

https://www.openprocessing.org/sketch/377698



Unsupervised learning

"Evolving Neural Networks through Augmenting Topologies"

by Kenneth O. Stanley and Risto Miikkulainen

Identifying weights and bias using Genetic Algorithm

Some application

https://www.youtube.com/watch?v=BBLJFYr7zB8

Unsupervised learning

"Evolving Neural Networks through Augmenting Topologies"

by Kenneth O. Stanley and Risto Miikkulainen

The idea is to find the optimal configuration of the network. Topology is variable

Unsupervised learning

“Neural Network Weight Selection Using Genetic Algorithms”

by David J. Montana

“Parameter Tuning of MLP Neural Network Using Genetic Algorithms”

By Meng Joo Er and Fan Liu

Topology is fixed, and the idea is to find the best weights and bias