7. The Evolution of Code

박종화
suakii@gmail.com

LECTURE PRESENTATIONS

For CAMPBELL BIOLOGY, NINTH EDITION

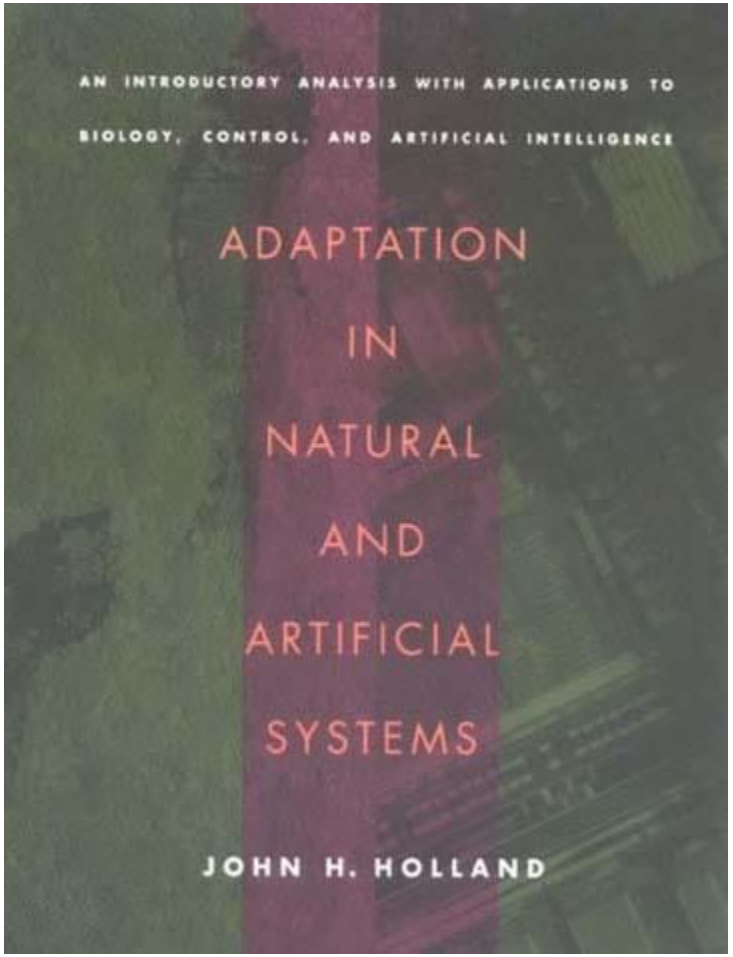
Jane B. Reece, Lisa A. Urry, Michael L. Cain, Steven A. Wasserman, Peter V. Minorsky, Robert B. Jackson

Lectures by
Erin Barley
Kathleen Fitzpatrick

Genetic Algorithms

- 진화의 모방이 아닌 진화의 활용
  - 고전적인 유전 알고리즘
  - 대화형 선택
  - 생태계 모방

History


AN INTRODUCTORY ANALYSIS WITH APPLICATIONS TO
BIOLOGY, CONTROL, AND ARTIFICIAL INTELLIGENCE
ADAPTATION
IN
NATURAL
AND
ARTIFICIAL
SYSTEMS
JOHN H. HOLLAND

History

- John Henry Holland (February 2, 1929 – August 9, 2015) was an American scientist and Professor of psychology and Professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. He was a pioneer in what became known as genetic algorithms.

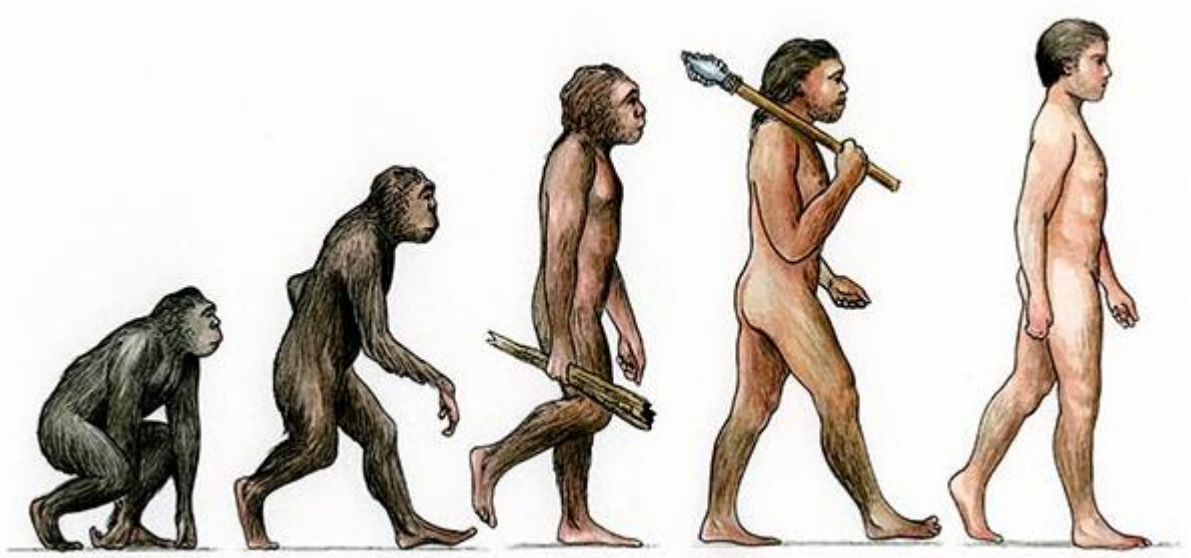
To be or not to be that is the question

$$\left(\frac{1}{27}\right)^{39}_{1}$$

$$= \frac{}{66,555,937,033,867,822,607,895,549,241,096,482,953,017,615,834,735,226,163}$$

우주의 나이를 제곱한 만큼의 시간이 필요!

Evolution of Code



pekmeobakdujepwkedainpobochoijeohgln

teabeghrfncvtombertfegdisbtfedquehjuon

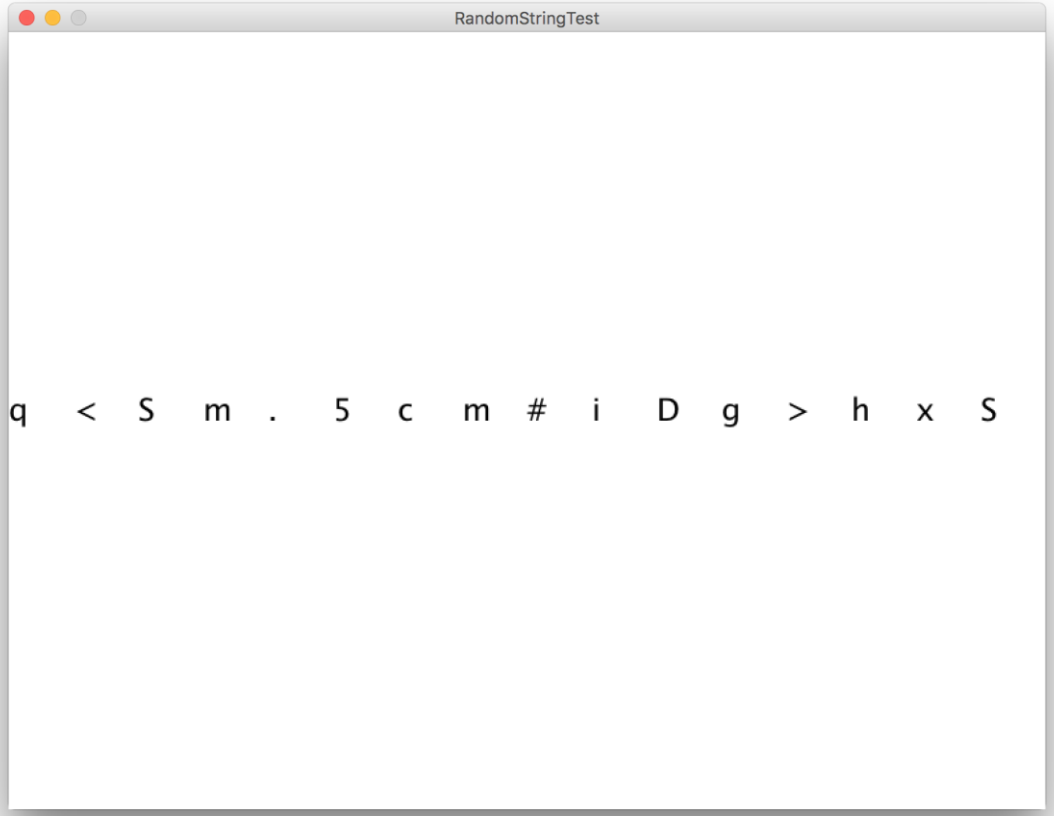
toabeghrfnotutomberthatdisbthedquestion

to be or not to be that is the question

Random Typing..

```
void setup(){
  size(800, 600);
  PFont f = createFont( "Arial-Black-24", 24 );
  textFont(f);
  background(255);
  frameRate(10);
}
void draw(){
  background(255);
  fill(0);
  for (int i = 0; i < width; i+=50) {
    text( "" + char( int( random(33,126) ) ), i, height/2 );
  }
}
```

Random Typing

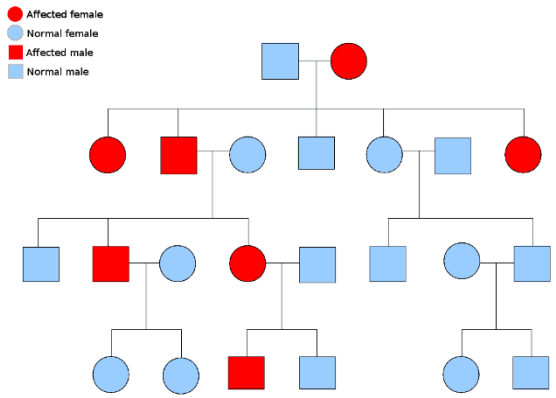


Darwinian Natural Selection

- ***Heredity***
- ***Variation***
- ***Selection***

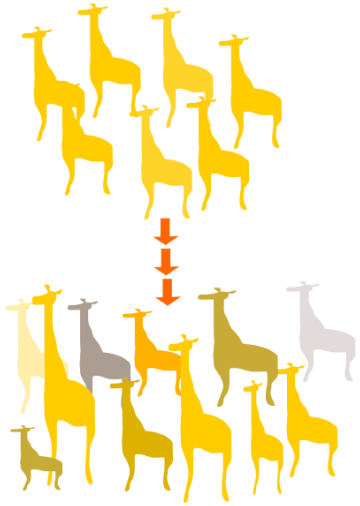
Darwinian Natural Selection

유전



자식에게
자신의 형질 전달

변이



다양한 형질 존재
+ 돌연변이 발생
새롭고 다양한 개체 형성

선택

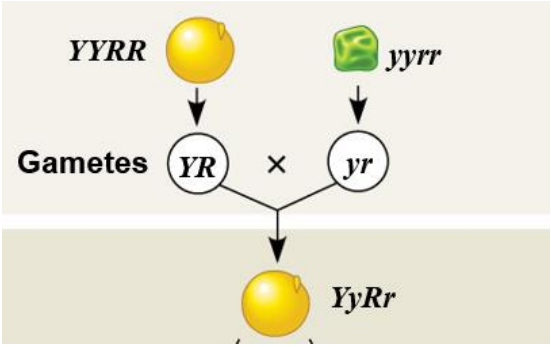
환경에 잘 적응한 객체만이
자식에게 형질 전달 가능

Part I: Creating a Population

```
String str = "cat";
```

랜덤하게 생성된
요소로 집단을 생성한다.

hut car box

유전자형		표현형
hut		hut
car		car
box		box
YY		
Yy		
yy		
0,0,0		
128,128,128		
255,255,255		
244,114,208		

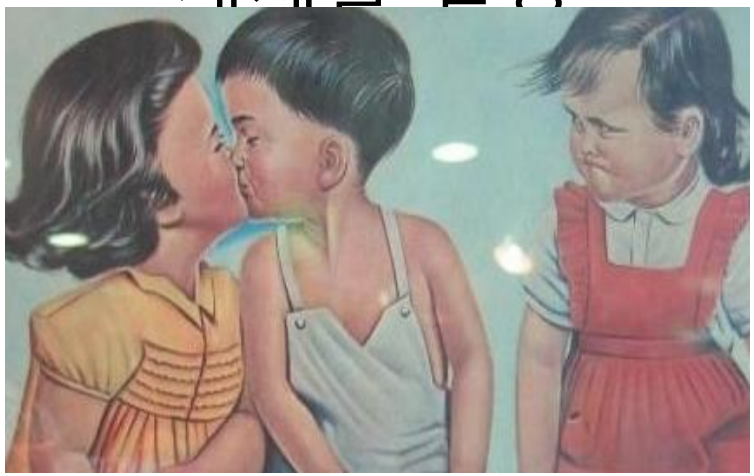
Part I: Creating a Population

- Create a population of N elements, each with randomly generated DNA.

- 집단의 생성은 변이법칙을 이용

Part II: Selection

- 다윈의 선택의 법칙
  - 집단을 선택하고 다음 세대의 부모로 적합한 개체를 결정



현실

집단에 적절해야 교배를 통해 자신의 형질 전달 가능

알고리즘

적응도 함수 결과가 좋아야 살아남아 형질 전달 가능

적응도 평가

DNA	Fitness
hut	1
car	2
box	0

교배풀 생성

방법 1. 엘리트 방식

적응도함수 점수가 높은 몇몇 개체만 교배 가능

→ 자식 세대의 다양성 문제 발생(변이 법칙 위배)

문제점 1. 계속 반복하면 자식 세대의 다양성이 점점 줆
문제점 2. 점수가 가장 높은 개체와 커트라인에 턱걸이한 개체의 번식률이 같음 → 비합리적!

교배풀 생성

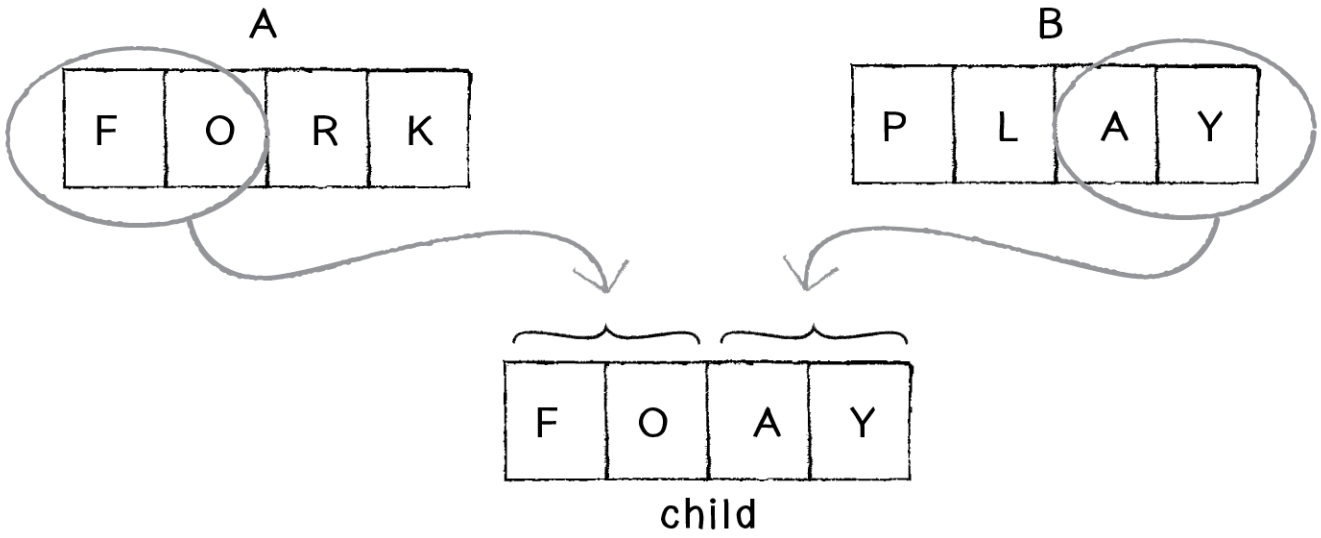
방법 2. 확률론 방식

적응도함수 점수에 비례한 교배 확률

→ 점수의 정규화(벡터의 정규화와 유사)

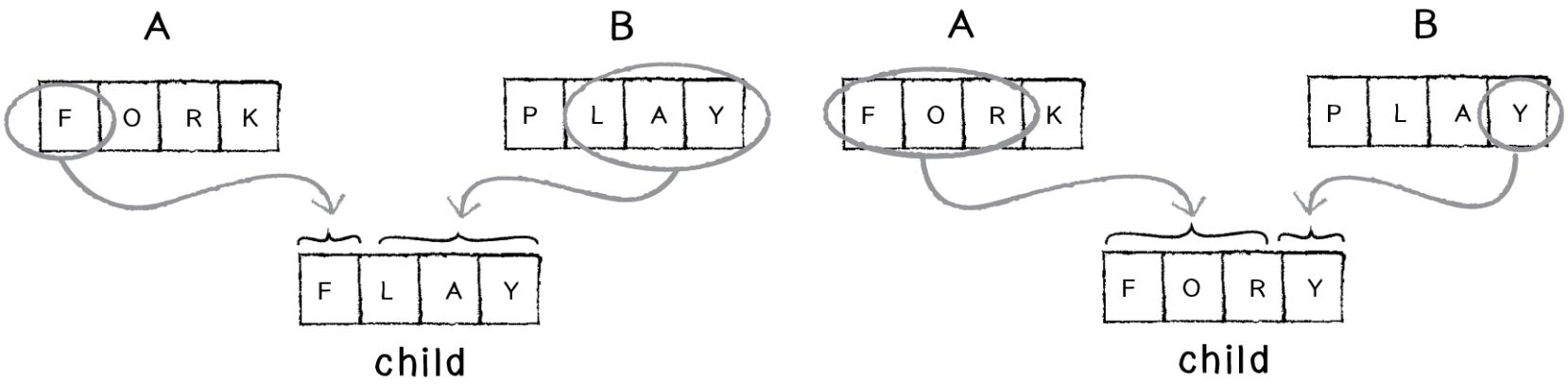
장점 1. 적응도가 높은 개체가 번식에 더 자주 참여
→ 우수한 형질 가진 자식 수 증가
장점 2. 자식 세대 개체의 유전적 다양성 확보 가능

Part III: Reproduction-Crossover



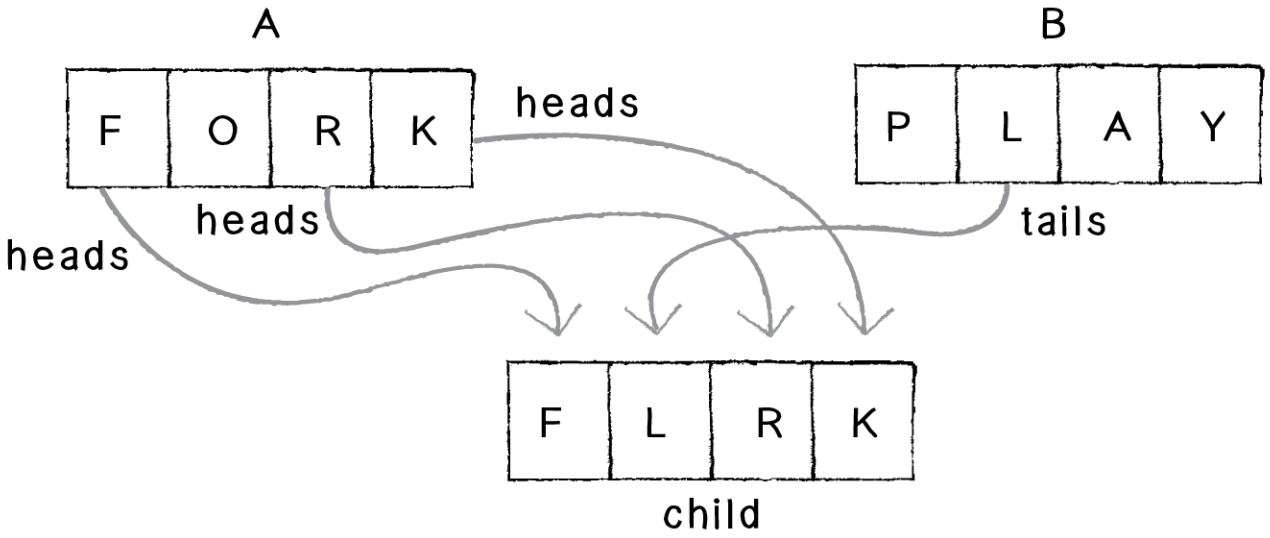
Part III: Reproduction-Crossover

Picking a random midpoint

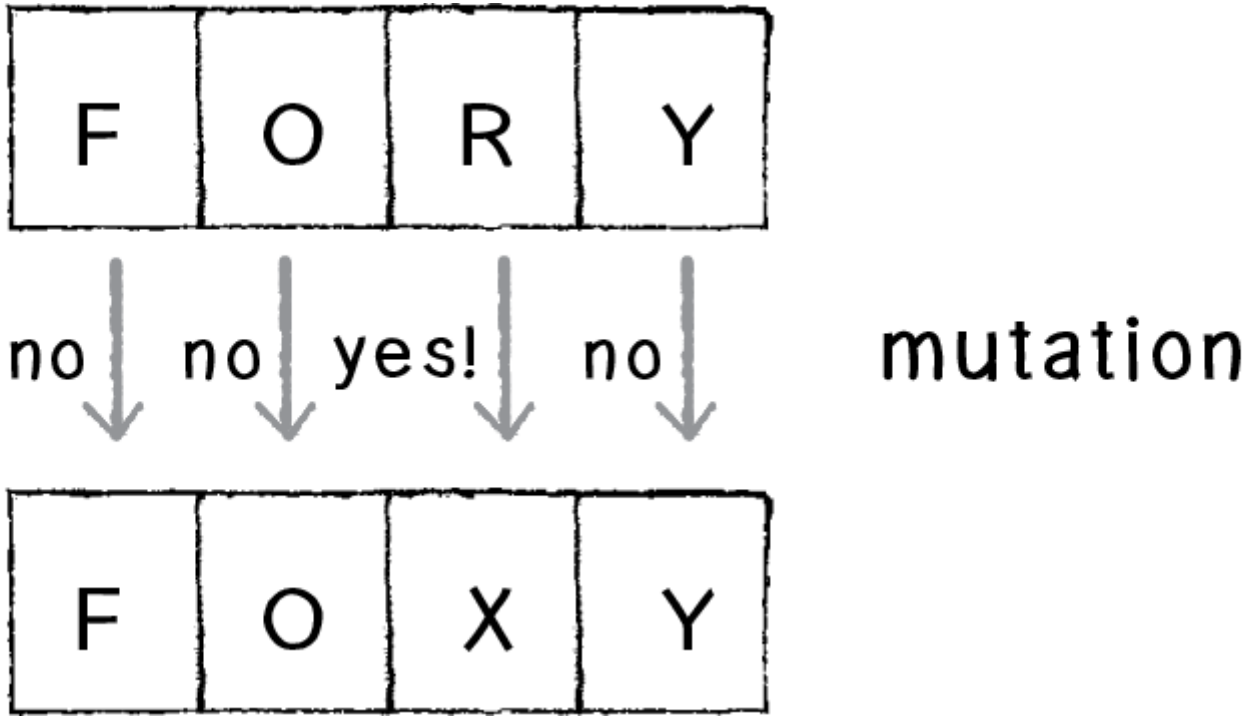


Part III: Reproduction-Crossover

Coin-flipping approach



Part III: Reproduction-Mutation



Part III: Reproduction

SETUP:

Step 1: ***Initialize***. Create a population of N elements, each with randomly generated DNA.

LOOP:

Step 2: ***Selection***. Evaluate the fitness of each element of the population and build a mating pool.

Step 3: ***Reproduction***. Repeat N times:

a) Pick two parents with probability according to relative fitness.

b) Crossover—create a "child" by combining the DNA of these two parents.

c) Mutation—mutate the child's DNA based on a given probability.

d) Add the new child to a new population.

Step 4. Replace the old population with the new population and return to Step 2.

Code for Creating the Population

- Step 1: 집단의 초기화
  - 집단에 속한 개체의 유전 정보를 저장하는 객체 - DNA

```
class DNA {
    }
```

The population will then be an array of DNA objects.

Code for Creating the Population

```
class DNA {
        char[] genes = new char[18];
}
```

Code for Creating the Population

```
class DNA {
  char[] genes = new char[18];

  DNA() {
    for (int i = 0; i < genes.length; i++) {
       genes[i] = (char) random(32,128);
    }
  }
}
```

Code for Creating the Population

- Step 2: 선택
  - 집단의 각 요소마다 적응도를 평가하고 교배 풀을 생성하는 단계

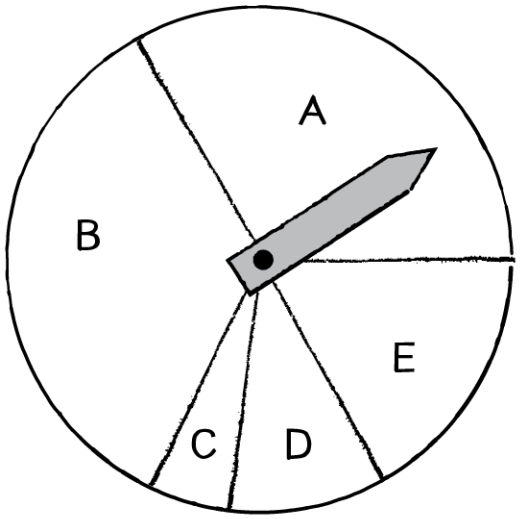
- **Fitness = Total # Characters Correct/Total # Characters**
-

Code for Creating the Population

```
class DNA {

void fitness () {
    int score = 0;
    for (int i = 0; i < genes.length; i++) {
        if (genes[i] == target.charAt(i)) {
                score++;
        }
    }
    fitness = float(score)/target.length();
  }
```

Code for Creating the Population

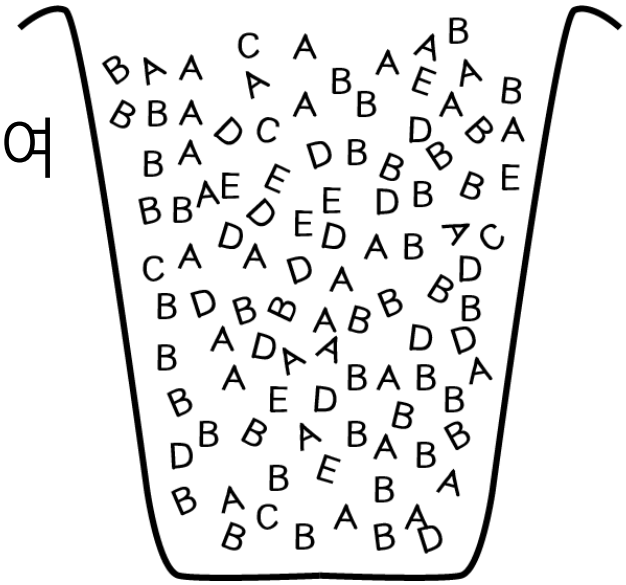


Spin the wheel!

Parent	Probability
A	30%
B	40%
C	5%
D	10%
E	15%

Code for Creating the Population

평가된 DNA의
적합도에 비례하여
교배 풀에
객체를 넣는다



Code for Creating the Population

```
ArrayList<DNA> matingPool = new ArrayList<DNA>();

  for (int i = 0; i < population.length; i++) {
      int n = int(population[i].fitness * 100);
      for (int j = 0; j < n; j++) {
          matingPool.add(population[i]);
      }
  }
```

Code for Creating the Population

- Step 3: Reproduction
- int a = int(random(matingPool.size()));
- int b = int(random(matingPool.size()));
- DNA parentA = matingPool.get(a);
- DNA parentB = matingPool.get(b);
- DNA child = parentA.crossover(parentB);
- child.mutate();

Code for Creating the Population

```
DNA crossover(DNA partner) {

DNA child = new DNA();

int midpoint = int(random(genes.length));

    for (int i = 0; i < genes.length; i++) {
if (i > midpoint) child.genes[i] = genes[i];
       else child.genes[i] = partner.genes[i];
    }

return child;
  }
```

Code for Creating the Population

```
void mutate() {
for (int i = 0; i < genes.length; i++) {
        if (random(1) < mutationRate) {
genes[i] = (char) random(32,128);
        }
    }
  }
```

Genetic Algorithms: 적용

- 첫 번째 변경 : 변수 변경

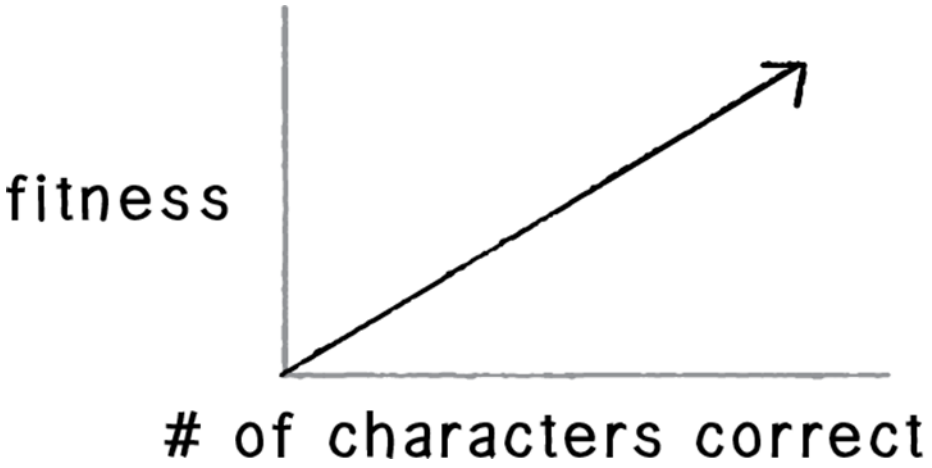
Total Population	Mutation Rate	Number of Generations until Phrase Solved	Total Time (in seconds) until Phrase Solved
150	1%	1089	18.8
300	1%	448	8.2
1,000	1%	71	1.8
50,000	1%	27	4.3

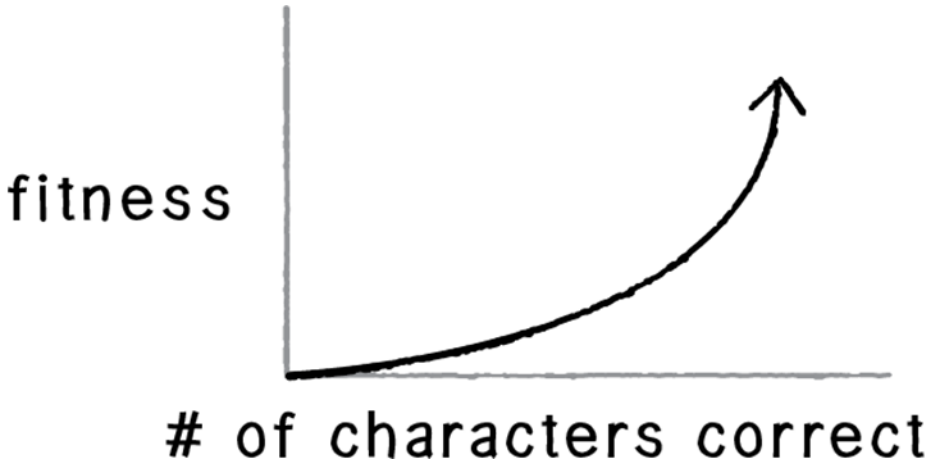
Genetic Algorithms: 적용

Total Population	Mutation Rate	Number of Generations until Phrase Solved	Total Time (in seconds) until Phrase Solved
1,000	0%	37 or never?	1.2 or never?
1,000	1%	71	1.8
1,000	2%	60	1.6
1,000	10%	never?	never?

Genetic Algorithms: 적용

- 적응도 함수





지수함수적인 평가함수는
더 좋은 개체를
많이 선택하게 해 준다

Genetic Algorithms: 적용

- 유전자형과 표현형

```
class Vehicle {
DNA dna;

  float maxspeed;
  float maxforce;
  float size;
  float separationWeight;

  Vehicle() {
    DNA = new DNA(4);
maxspeed = dna.genes[0];
    maxforce = dna.genes[1];
    size = dna.genes[2];
    separationWeight = dna.genes[3];
}
```

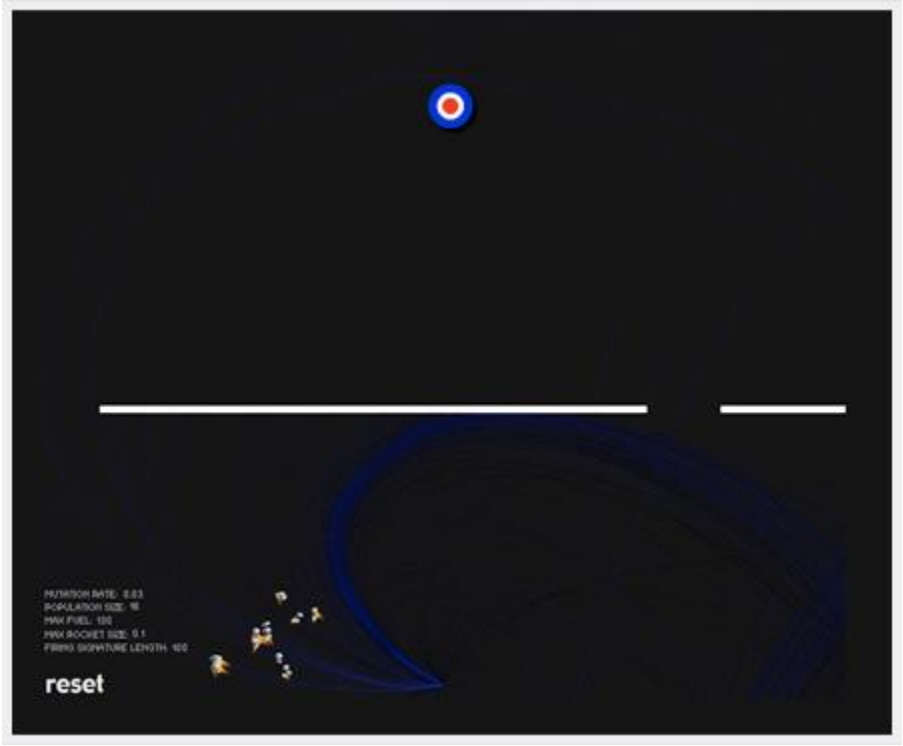
DNA를 일반화하고,
표현형에서 형질을
발현한다

유전 알고리즘 응용: Smart Rocket

Smart Rocket

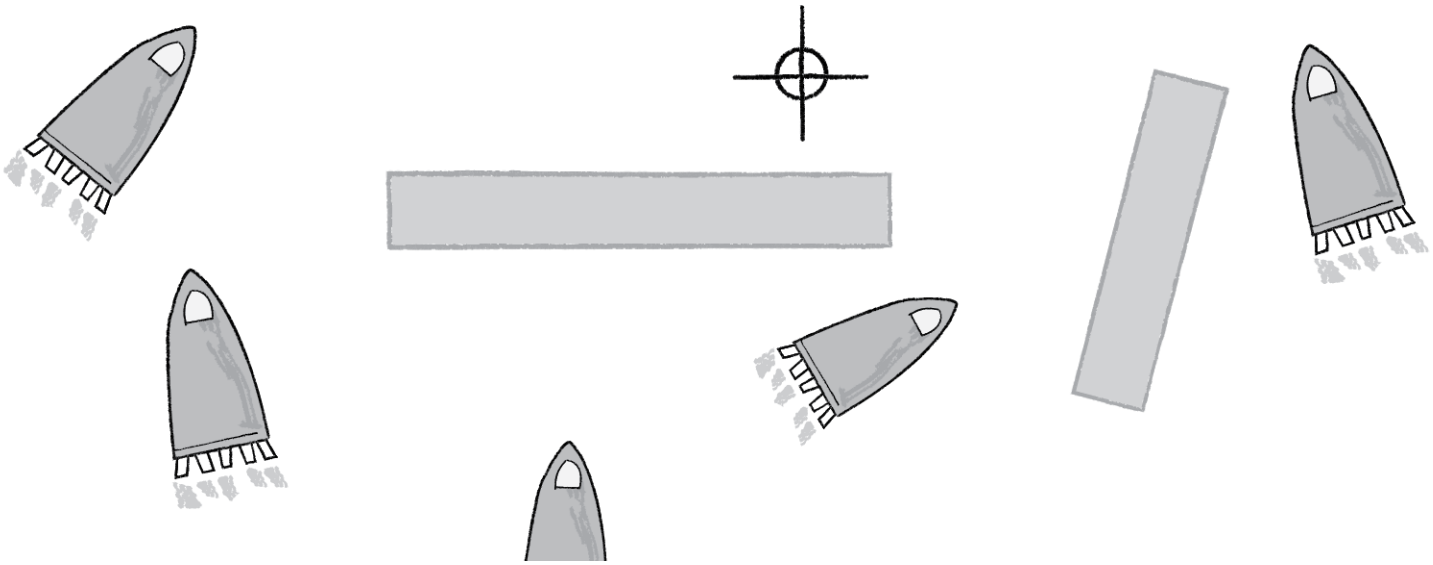
http://blog.blprnt.com/blog/blprnt/project-smart-rockets
2009

Smart Rocket



Evolved antenna

Smart Rocket



타겟을 향하는 로켓을 만들자

Smart Rocket

이런 움직이는 객체에는

만능 클래스 Mover를 사용한다

Smart Rocket

```
class Rocket {
  PVector location;
  PVector velocity;
  PVector acceleration;
  void applyForce(PVector f) {
    acceleration.add(f);
  }
  void update() {
    velocity.add(acceleration);
    location.add(velocity);
    acceleration.mult(0);
  }
}
```

Rocket Class의 원형

Smart Rocket 1

- 집단의 요소 수와 돌연변이율
- -로켓 100기
- -돌연변이율 1%

Smart Rocket 2

$$fitness \propto \frac{1}{distance}$$

적합도는 타겟과의 거리에 반비례한다

Smart Rocket

```
void fitness() {
  float d = PVector.dist(location,target);
  fitness = 1/d;
}
```

Smart Rocket

- 유전자를 만들자.

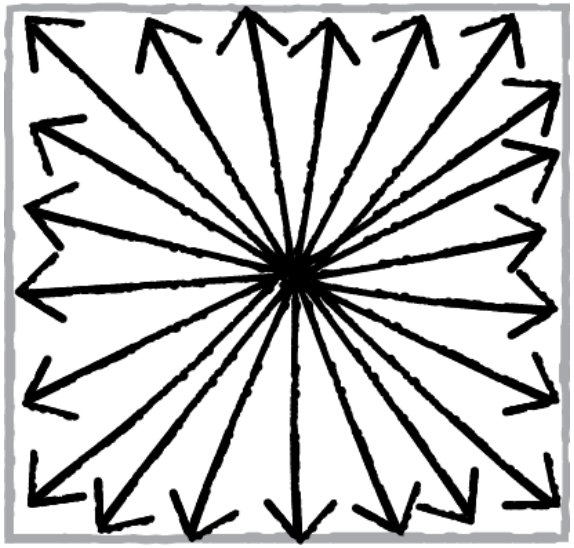
Smart Rocket

- 로켓에는 프레임마다 크기와 방향을 자유롭게 변형할 수 있는 추진 엔진이 존재

- DNA는 로켓의 변위 벡터가 된다.

Smart Rocket

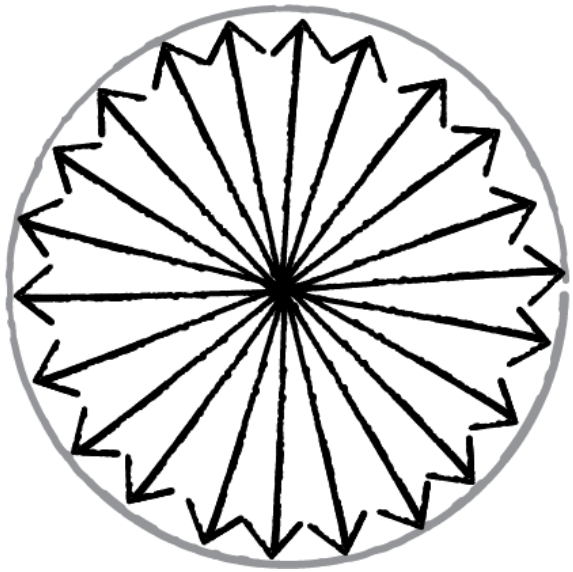
```
PVector v = new PVector(random(-1,1),random(-1,1));
```



유전자의 편향이 일어난다

Smart Rocket

PVector v = PVector.random2D();



유전자의 편향을 없애기 위해서는
원 모양의 랜덤 벡터를 생성한다

Smart Rocket

```
class DNA {
  PVector[] genes;
  float maxforce = 0.1;
  DNA() {
    genes = new PVector[lifetime];
    for (int i = 0; i < genes.length; i++) {
      genes[i] = PVector.random2D();
      genes[i].mult(random(0, maxforce));
    }
  }
}
```

Smart Rocket

```
class Rocket {
  DNA dna;
  float fitness;
  PVector location;
  PVector velocity;
  PVector acceleration;
  int geneCounter = 0;
  void run() {
    applyForce(dna.genes[geneCounter]);
    geneCounter++;
    update();
  }
}
```

Smart Rocket

```
class Population {

float mutationRate;
  Rocket[] population;
  ArrayList<Rocket> matingPool;
  int generations;

void fitness() {}
  void selection() {}
  void reproduction() {}
}
```