Evolutionary Programming in Artificial Intelligence.

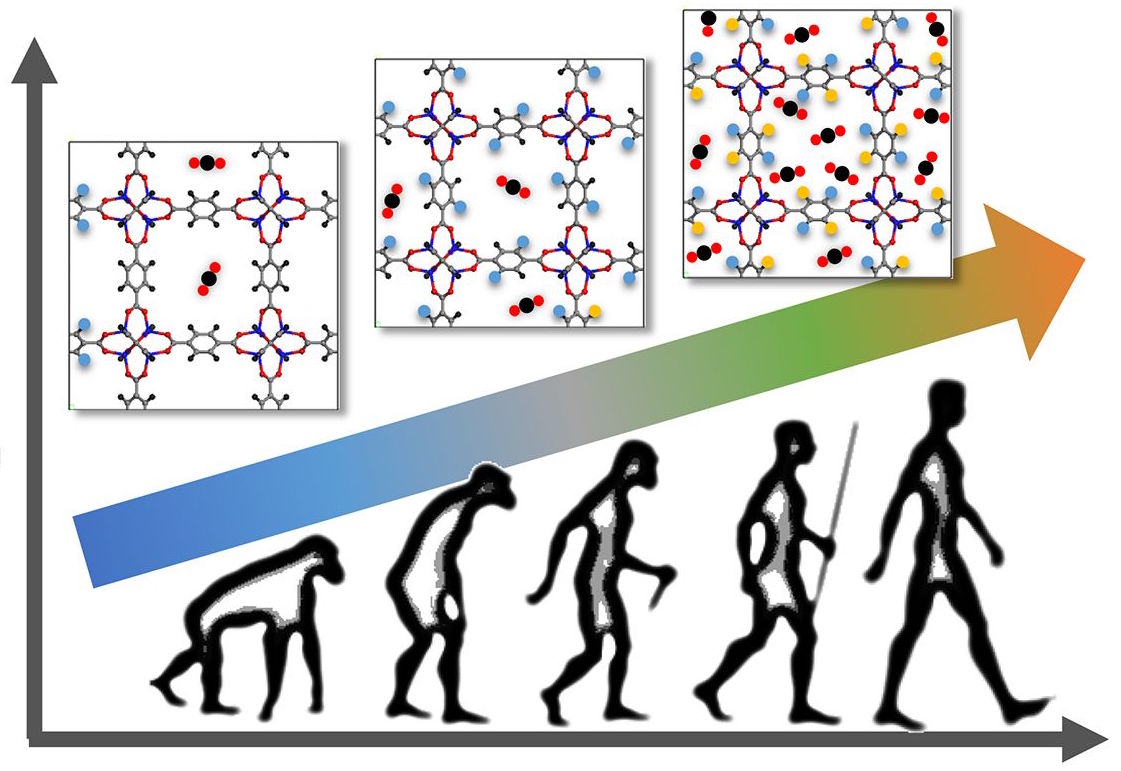
A study focusing on Genetic Algorithms.

By Aahil Awatramani

What are Genetic Algorithms?

Genetic algorithms are a set of rules followed by a computer which are backed up by the biological model of evolution and natural selection, introduced first by Charles Darwin in 1859. In our structured world, evolution helps species adapt to their environments.

Genetic Algorithms deal with finding solutions to problems. Over time the quality of the solutions improves. If algorithm is successful, we can terminate the evolution once it has found a solution that is good enough.

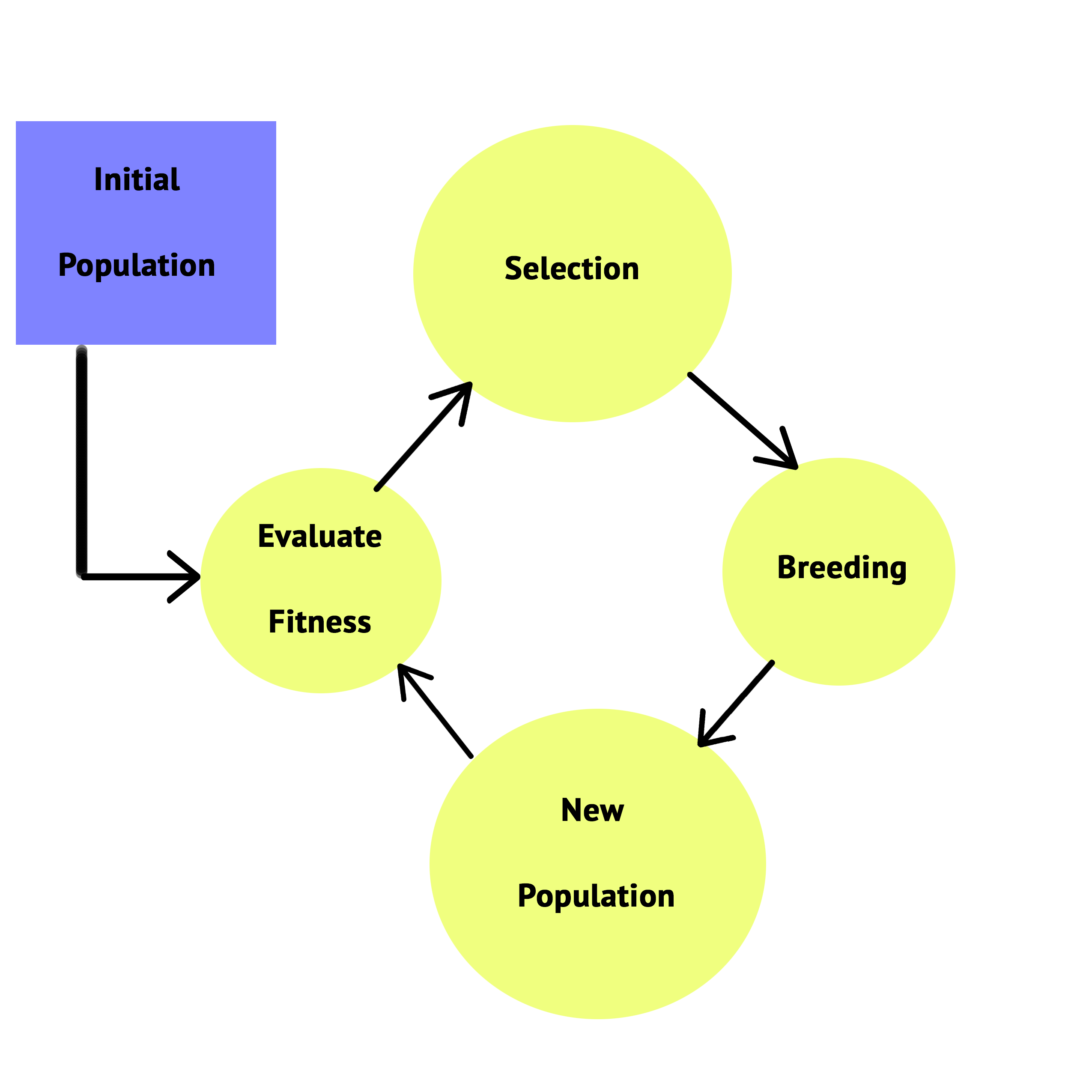


Genetic Algorithms work best based on the phrase “Survival of the Fittest”, granted that there is still a certain random aspect to it. The fittest have the highest chance of survival while the weakest have the lowest.

Genetic Algorithms occur in 4 Phases:

1. Initial Population, which sets a ‘n’ sized population of genomes with random genes each.
2. Evaluation of Fitness Level, which determines how close each individual genome is to the solution.
3. Selection, which picks and chooses the fittest genome for breeding an offspring.
4. Breeding, which breeds a newer and improved population from the older population.

This process is repeated like a cycle until a suitable outcome is observed.



The important parts of a Genetic Algorithm lie in fitness evaluation step, the Selection Step and the Breeding step.

1)Fitness Evaluation Function:

Factors that influence the survival prospects of an individual help measure its fitness. Fitness function helps convey how close a genome is to its end goal. A good fitness function should be able to return distinct fitness levels which can differentiate between a strong genome and a weak genome. Fitness functions are able to do this by returning scores for each genome for example a child who got a score of 90 on a test will clearly be more suited to that system than a child who got 30 on the same test. The probability of being chosen for reproduction is based on your fitness level.

The algorithm evaluating the fitness of a genome depends on your program. Each genome is subjected to an evaluation test which will return a score based on how close it is to the end result. The closer it is the higher the fitness will be.

2)Selection Function:

Two genomes are selected at random for reproduction. The selection is aided by the fitness score giving the fittest genomes a higher chance of selection.

Different selection algorithms may result in slower or faster evolution. Depending on the program, finding the right selection algorithm contributes to a major aspect of its efficiency.

Selection Algorithms:

1. Steady State Selection:

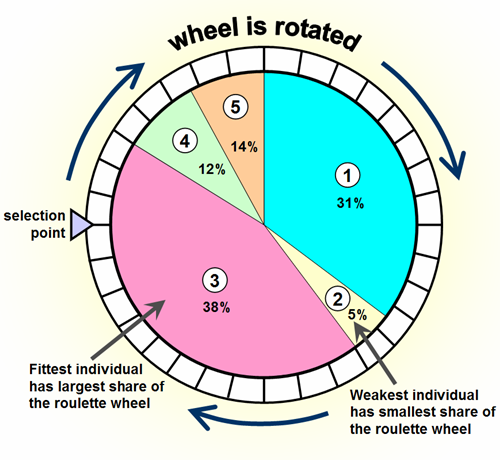
This method is also known as Elitism as it keeps the genomes of the highest fitness and uses them to breed offspring that will replace genomes of lower fitness.

Elitism can rapidly increase the performance of the program as it prevents the loss of the best found solutions and can even work with negative fitness values.

The algorithm for Steady State Selection is:

1. Arranging all the genomes in order of their fitness that is from Highest Fitness to Lowest Fitness.
2. Deleting half of the genomes with lower fitness scores.
3. Replacing all the dead genomes with the offspring of all the surviving genomes. Two offspring per mother and father pair in order to maintain the population size.
4. Roulette Wheel Selection:

This selection process is to stochastically select genomes, where the fittest have a greater probability of being picked. However, the weakest are not without a chance.

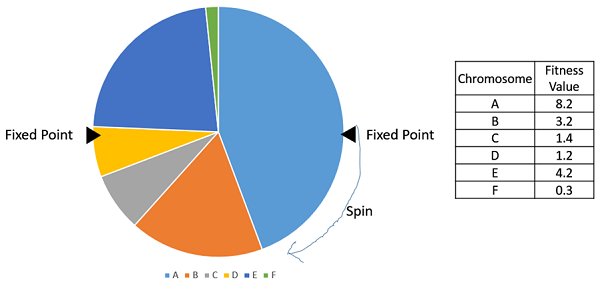


It is clear that a fitter genome has a greater section on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated.

The algorithm for Roulette Wheel Selection is:

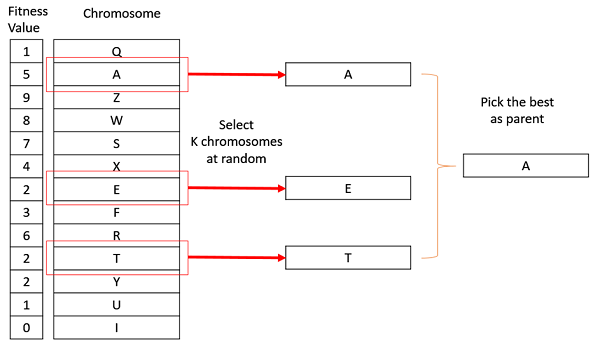
1. Calculate S = the total fitness value.
2. Generate a random number between 0 and S.
3. Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
4. The genome for which P > S is the chosen genome.
5. Stochastic Universal Sampling:

Stochastic Universal Sampling is similar to Roulette Wheel Selection, however instead of having just one selection point there are multiple selection points. Therefore, all parents are chosen in a single spin of the Wheel.



1. Tournament Selection:

In a N-Way tournament selection, N number of genomes are selected from the population at random and the best out of these genomes is selected to become a parent. The process is repeated for selection the next parent as well. The Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



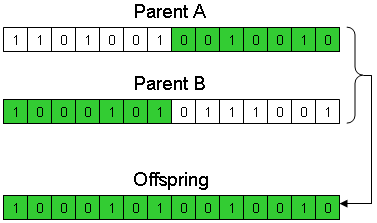
1. Random Selection:

In this method parents are simply chosen randomly from the existing population. There is no selection pressure as fitness is not taken into consideration at all, and therefore this method should be avoided.

3)Breeding Function:

The breeding function consists of 2 steps:

The Crossover: For each pair to be mated a crossover point is chosen at random from a bit string. Offspring are created by exchange of genetic material between the parents at the cross over point. The population is generally diverse in the beginning hence the crossovers have a large impact in the beginning however towards the higher generations they settle down and have distinct smaller impacts.



The Mutation: Mutation adds the random element to evolution. Without mutation we would never be able to adapt to newer environments and hence we would soon become extinct. It simply adds a random factor by either changing one bit in the bit string or even randomizing all of it.

When are Genetic Algorithms useful?

Genetic Algorithms are typically used to provide decent solutions to problems that cannot be easily dealt with using traditional techniques. Several optimization tasks fall into this category. It may be too hard to find an exact solution but sometimes a near-perfect solution is sufficient. In these circumstances, evolutionary algorithms can be used. Due to their unpredictable nature, genetic algorithms are never guaranteed to find a perfect answer for any problem, but they will often find a good solution if one exists.

One example of this kind is the challenge of scheduling. Schools must arrange room and staff divisions to suit the needs of their schedule. There are many constraints that must be fulfilled. A member of staff can only be in one place at a time, they can only teach classes that are in their area of expertise and rooms cannot host lessons if they are already occupied. It is not practical to comprehensively search for the ideal timetable due to the massive amount of calculation involved. Genetic algorithms have proven to be a successful way of generating satisfactory solutions to many scheduling problems.

My Own Projects on Genetic Algorithms:

1. Binary Number Duplication:

Level: Beginner

Language: Java

Platform: BlueJ

Aim: To reach a target sequence of 1s and 0s through genetic algorithm programming.

Algorithm:

1. Create Initial Population of 8 genomes with a randomized bits sequence, example genome [0] = {1,0,1,1}, and store them in an object array.
2. Assign each genome a fitness score based on how close it is to the target genome, example if target genome is {1,0,1,0} then genome [0] has a fitness of 3.
3. Sort genomes in the object array according to their fitness. Fittest at the top and weakest at the bottom.
4. Breed the fittest genomes and replace the weaker genomes with the babies of the fit genomes.
5. Within the breeding method, create a mutating method which randomizes a baby’s bits’ sequence with respect to a low mutation rate.
6. Repeat process till desired outcome.

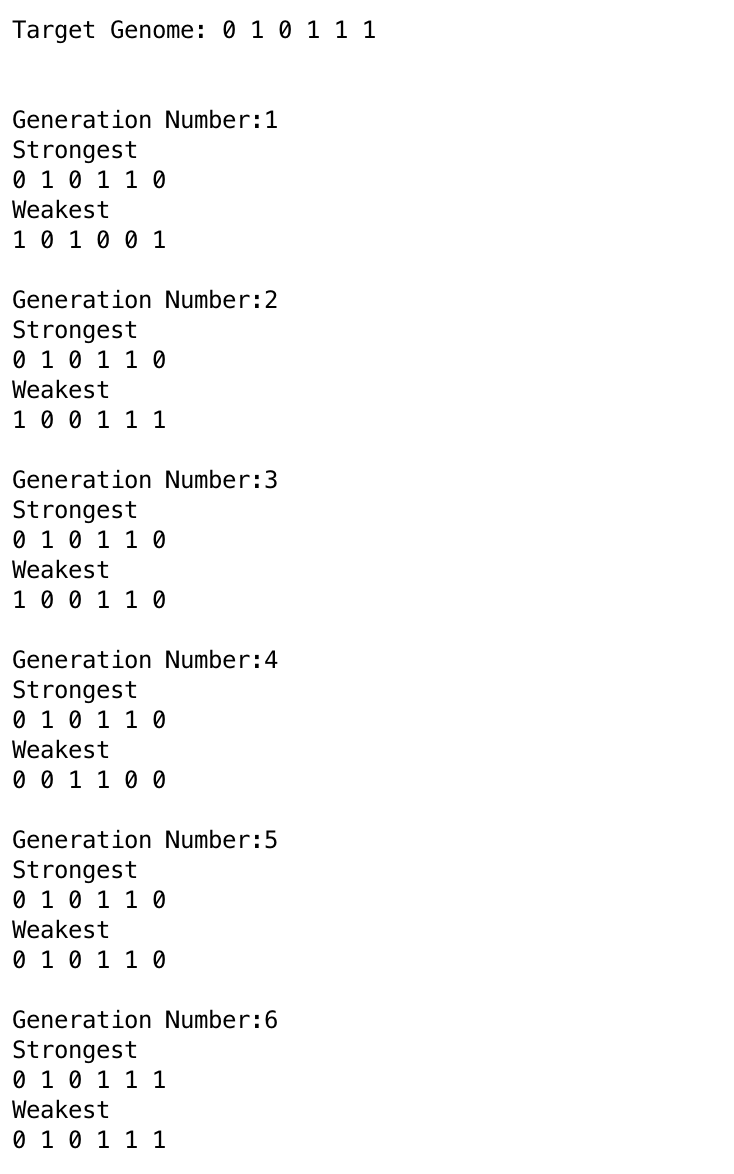
Notes and Difficulties:

This was my first attempt at programming a genetic algorithm hence I used a language that I am most comfortable with.

Despite being such a basic program I found the experience exceptionally stimulating.

It was something I had never programmed before in my life.

Output:



1. Color Survive:

Level: Medium

Language: C#

Platform: Unity

Aim: To make the colors of capsules adapt to a periodically changing background color through genetic algorithms.

Algorithm:

1. Create a plane in which the color changes periodically.
2. Create an Initial population of 100 random colored capsules into a list and place them on the plane.
3. Assign a fitness to each capsule, by subtracting the Vector form of their colors from the Vector form of the current background color and return a magnitude from this.
4. Sort the capsules according to the fitness, closer the value is to 0, higher the fitness is. Fittest capsules are sorted at the top of the list and weakest capsules are kept at the bottom.
5. Create a halfway mark which is at half the size of the population. Delete all capsules below the halfway mark in the sorted list.
6. Breed the remaining capsules amongst the fittest, creating two babies for each parent pair. Add the babies to the list of capsules.
7. Within the breeding method create a mutating method which randomizes the color of the capsule with respect to a low mutation rate.
8. Repeat process and observe outcomes.

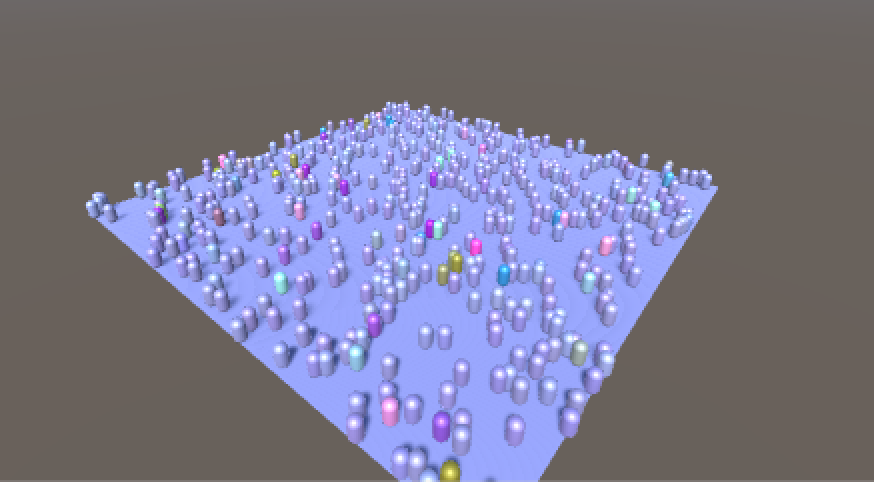
Notes and Difficulties:

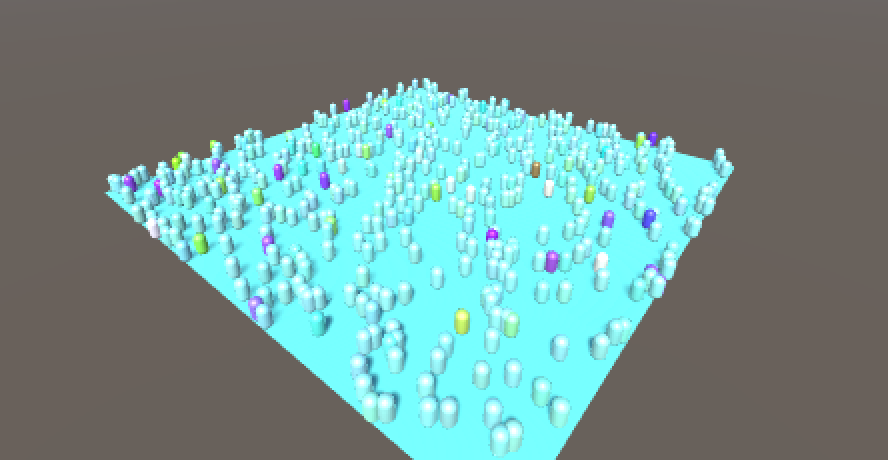
Since I had thoroughly understood the methodology behind programming basic genetic algorithms I decided to program simulation of an example of “The Survival of the Fittest” using Unity engine. I had learnt C# programming on Unity Engine by myself during my summer break in the 11th Grade.

I struggled in programming this as my fundamentals in C# was not as strong as I had believed. I had overlooked one crucial step on giving each genome a behavior of color. I had instead set their colors by using a brute force method in my code as a shortcut. This backfired as each color was setting itself back to its default color whenever I would call upon that specific genome.

This taught me the importance of strengthening the fundamentals of a language before moving on to more advance coding. It also taught me that brute force should always be treated as a last resort only.

Output:





1. Maze Solver:

Level: Hard

Language: C#

Platform: Unity

Aim: To solve of maze through genetic algorithm programming.

Algorithm:

1. Create a maze of blocks using cubes with a distinct start and end.
2. Create an initial population of 150 genomes each containing a randomized bit sequence of 1s and 0s of length 70. Add these genomes to a list of genomes.
3. Assign a fitness to each genome by testing how far it made it to the finish line in the maze. This is done by
4. Decoding the bit string into integer values in the range 0 to 3.
5. Each integer value represents a cardinal direction in which the blocks will be placed.
6. If a block overlaps a wall it will not be placed.

The fittest genome sequence is noted for display purposes.

1. For Selection, pick a mother and a father genome using Roulette Wheel Selection.
2. Breed the selected mother and father genome to form two new babies and the babies to a list of genomes.
3. Within the breeding method, create a mutating method which flips a baby’s bits’ sequence (that is replacing all 1s with 0s and 0s with 1s) with respect to a low mutation rate.
4. Keep breeding until the number of babies is equal to the population size and then proceed to replace the previous genome list with the new one.
5. Repeat process until the maze is solved.

Notes:

I decided to challenge myself with an even harder program that I had noticed on a popular online forum.

I had received the idea of storing directions for the chain of blocks in the form of cardinal directions consisting of binary numbers from my AP Physics teacher, Mr. Mehra.

I was unable to use the same selection method I was used to (that is Elitism), as this method gets rid of weaker methods. In this program a fit chain of blocks, that is one close to the exit, could also be leading to a dead end. If a dead end strikes, then the program will be unable to continue as all the genomes will be bread from the ones blocked by the dead end. Hence for this program I used the Roulette Wheel Selection Method as it gives a chance to even the weaker genomes to breed, therefore the program may get a chance to start back at an earlier less fit checkpoint.

Output:



To Download the source file as well as the application files of Color Survive and Maze Solver Please click the given link below:

<https://drive.google.com/open?id=1Xl9Ks8poe09QLhBOz2SSQCapQz3kw_TE>

Bibliography:

Images:

https://phys.org/news/2016-11-genetic-algorithm-carbon-dioxide-sponge.html

http://mnemstudio.org/genetic-algorithms-recombination.htm

Books:

Buckland, Mat. *AI Techniques for Game Programming*. Premier Press, 2002.

Norvig, Peter, and Stuart J Russell. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, 2013.

Websites:

https://watchmaker.uncommons.org/manual/ch01s02.html

https://www.tutorialspoint.com/genetic\_algorithms/genetic\_algorithms\_parent\_selection.htm

Acknowledgements:

Special Thanks to Noureddin Sadawi, for answering all my doubts on Genetic Algorithms via email.