# CIS 412 Artificial Intelligence
# Project 1 Evolutionary Computation

**TEAM MEMBERS:**

1. **Vishnu Vardhan Kumar Pallati**
2. **Nicholas Elliott**
3. **Shrinivass Arunachalam balasubramanian**

## PART - 1

## Problem Statement

**Simulation of trajectories that are used to launch satellites from earth to moon - using genetic algorithm:**

Usually, calculating the trajectories and launch points for launching any satellite manually is difficult. In this project, we simulate the trajectories for launching the satellites by simplifying the calculation of these points using GA. Genetic Algorithm helps find trivial parameters that are used in calculation of trajectories in launching the satellites from earth to moon.

The trajectories are calculated using $\theta, V, \alpha$ *parameters*; where,

$\alpha$ is a tangential angle representing the angle from surface of earth at which satellite is launching,

$\theta$ represents the launch point of satellites i.e,. angle from center of earth to the point at which the satellite is launching and

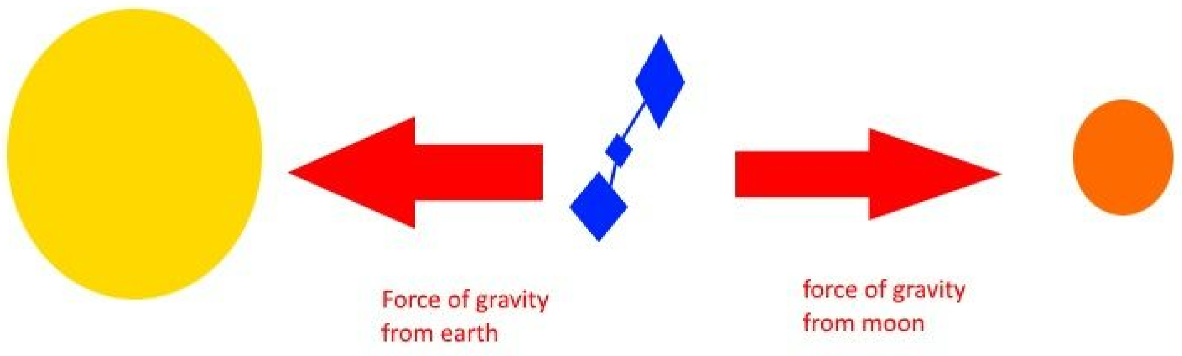
V represents the Initial speed at which the satellite launched.

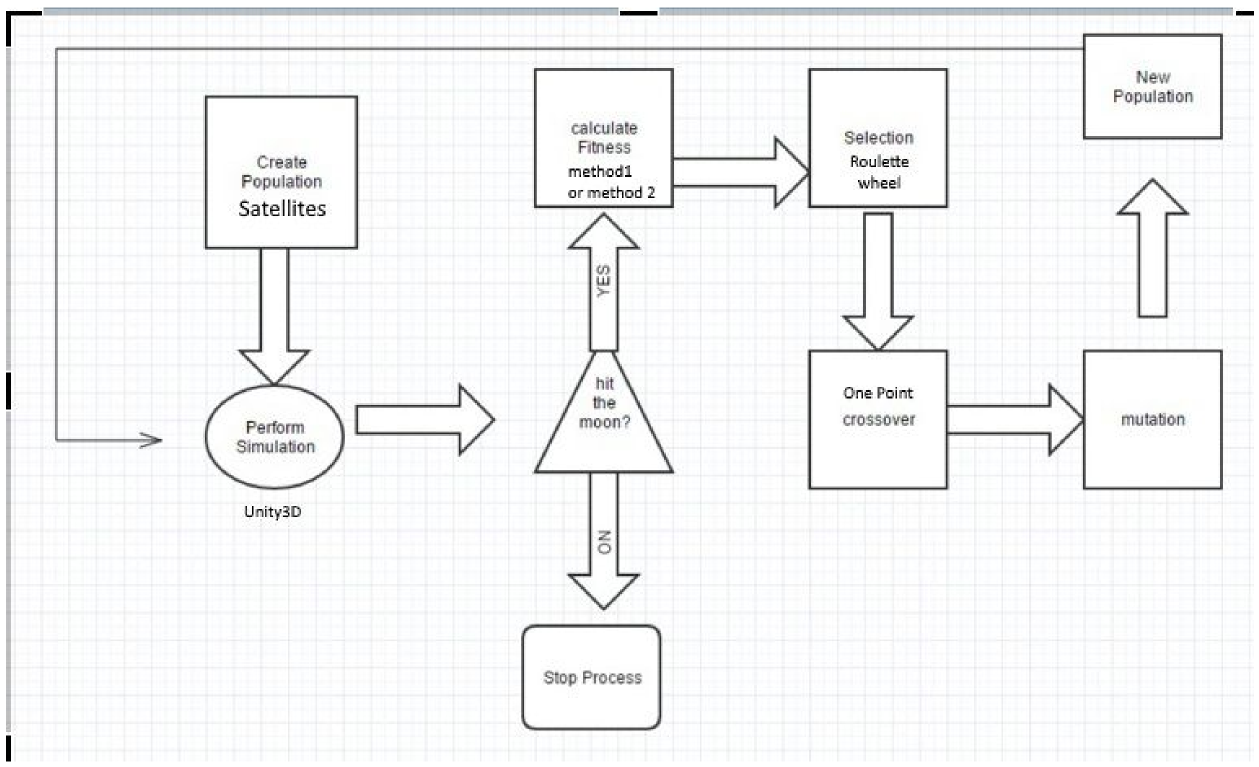
## Application of Genetic Algorithm

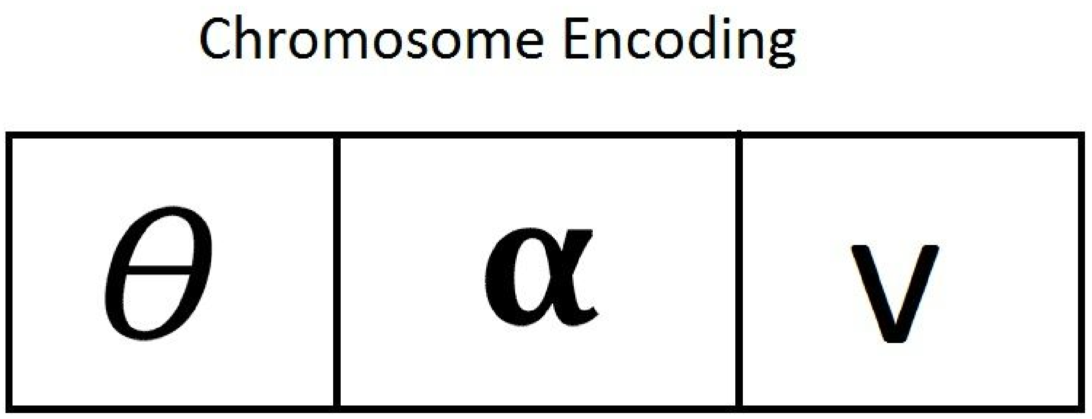
Initially, we are build a population of satellites with random values of $\theta$, $V$, $\alpha$ *parameters*. Fitness of each satellite is calculated according to the minimum distance ever reached in its journey. A new generation is created by crossing over the properties of previous generation. The satellites with higher fitness have a greater chance of participating in the cross over. Eventually the child generation will have higher fitness. By introducing an amount of mutation to the new generation the satellites with same properties would not repeat from generation to generation.

### Representation of Solution

We have developed a 2D simulation world in Unity3d game engine. The simulation contains a model of earth, a model of moon and a satellite prefab. The models are resized and positioned taking in consideration the real world dimensions and mapping each of them accordingly. The moon will be revolving around the earth and the satellites are launched with an initial speed. Unity physics engine takes care of adding the forces of gravity on the satellite from earth and moon and updates it every frame. The satellite simulates in the given simulated conditions and records its minimum distance ever recorded from moon in its journey and this can be used for fitness calculation.

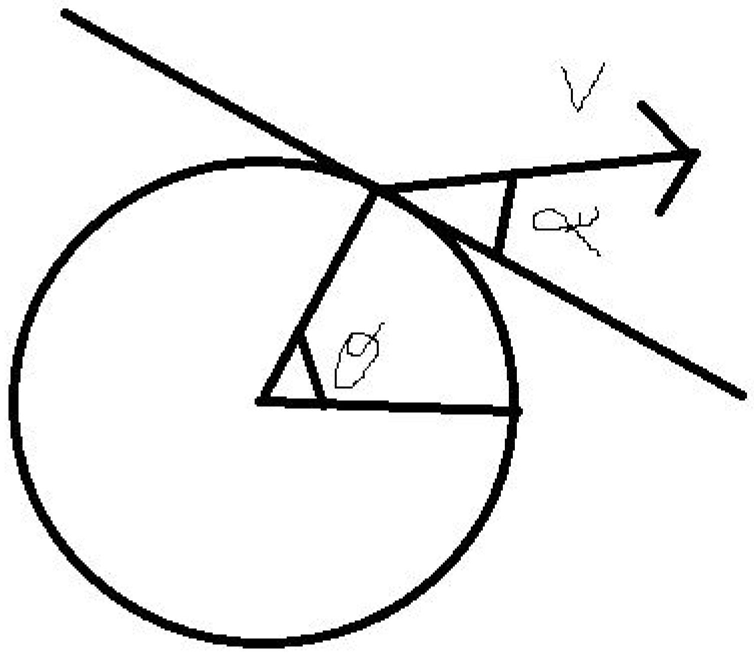






Each Chromosome represents a satellite. It is encoded with three variables $\theta, V, \alpha$ .

These three variables represent :



By implementing Genetic Algorithm to our proposed system we get a population of satellites which have higher chance to hit the moon.

**Building the Fitness Function**

We used two methods for finding the fitness

Method -1 : Representing the fitness as minimum distance from the moon ever reached by a satellite in its journey. Fitness ranges from 0 to 1000; where 0 is mapped to the maximum distance and 1000 maps to the satellites reaching moon.
Method - 2 : Representing the fitness as the sum of  minimum distance from the moon ever reached by a satellite in its journey with the time of journey. Fitness calculated from the distance ranges from 0-1000  and is mapped as specified in method 1 and the fitness calculated from the journey time goes from 0-200 ( The limit 200 can be changed dynamically using the fitness rate change factor).

**Selection of Individuals**

We perform a roulette wheel selection to select two satellites for cross over. For providing the satellites of greater fitness with a higher possibility of being selected, the roulette wheel selection was opted.

**Method of Crossover**

We perform a One-Point crossover at random points. The slice point is selected randomly for each crossover. As, there are only three parts in a chromosomes performing One-Point or Two-Point or Three-Point cross over would result in similar cross over. Thus, selecting any one of these would not make much of a difference.

**Implementation of Mutation**

Two types of mutations were implemented that are completely dependent on the mutation rate which could be changed by the user before simulation. In the first type of mutation we randomly change the $\theta$, $\alpha$, $V$ values of the satellite. For every 5% of mutation rate one satellite undergoes total randomness while in the other type of mutation we randomly select values of $\theta$, $\alpha$, $V$ from the range of its + or - initial values. For every 2% mutation rate a satellite is selected in this mutation type. The number of satellites selected for mutation also depends on the size of population. The above statistics are for a population with size 50.

**Termination criterion**

Termination criteria is reached when a satellite reaches the moon. This is the point where we stop searching and producing new generation.

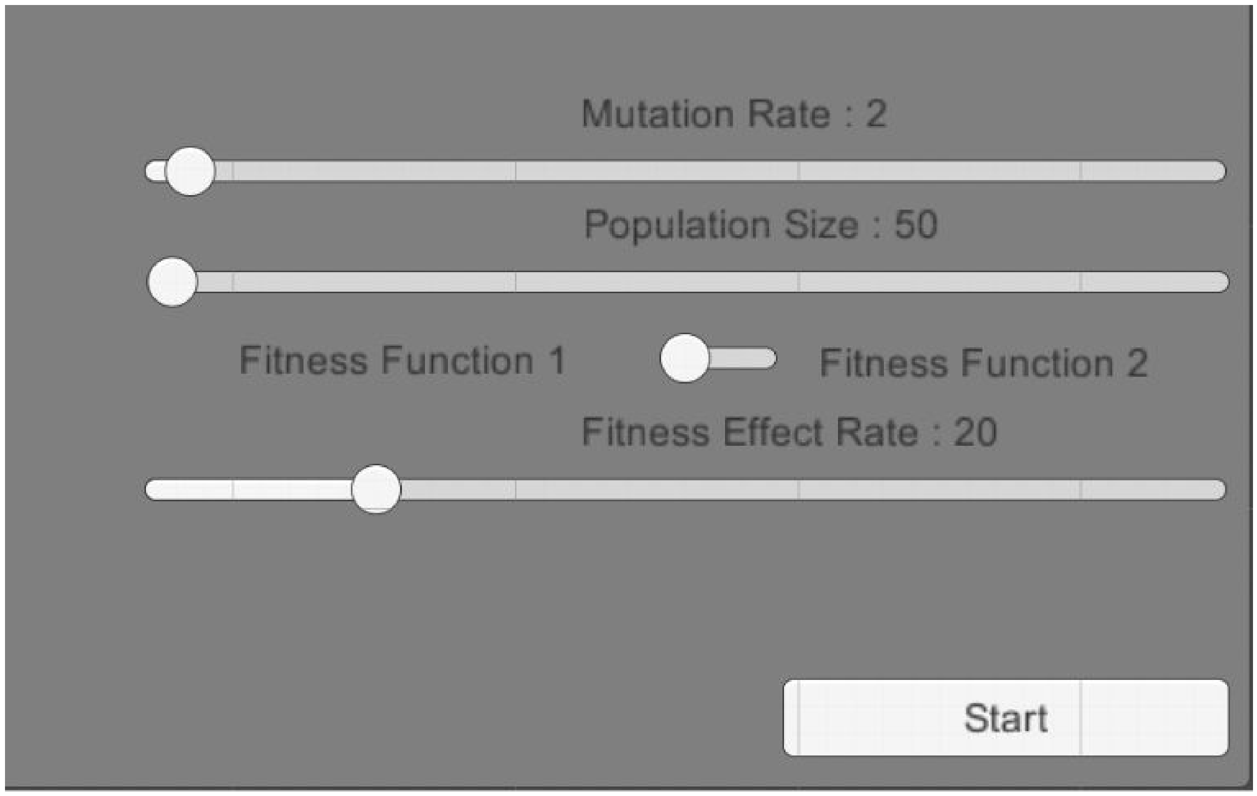
**Algorithm Reference:**
http://www.obitko.com/tutorials/genetic-algorithms/

**PART - 2**

**Source Code**

**The script files are attached in the zip file whereas the complete source code is a large file size and is available in Google Drive. It can be accessed using the following link.**
[https://drive.google.com/folderview?id=0BxsRsOT-45ZTMWx4Wl9JWnRRUWM&usp=sharing](https://drive.google.com/folderview?id=0BxsRsOT-45ZTMWx4Wl9JWnRRUWM&usp=sharing)

**The GUI Screen**



Mutation Rate : 2

Population Size : 50

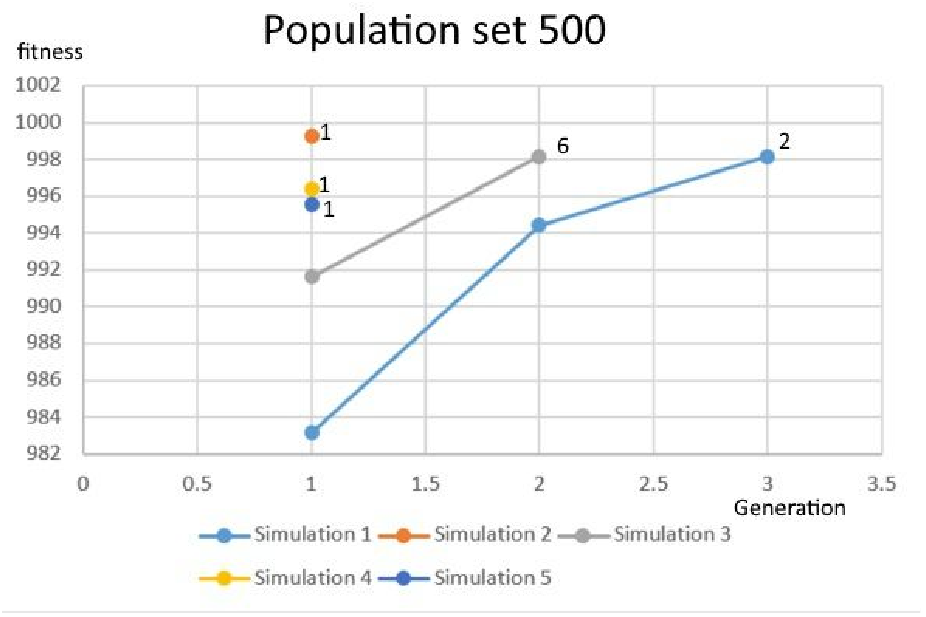
Fitness Effect Rate : 20

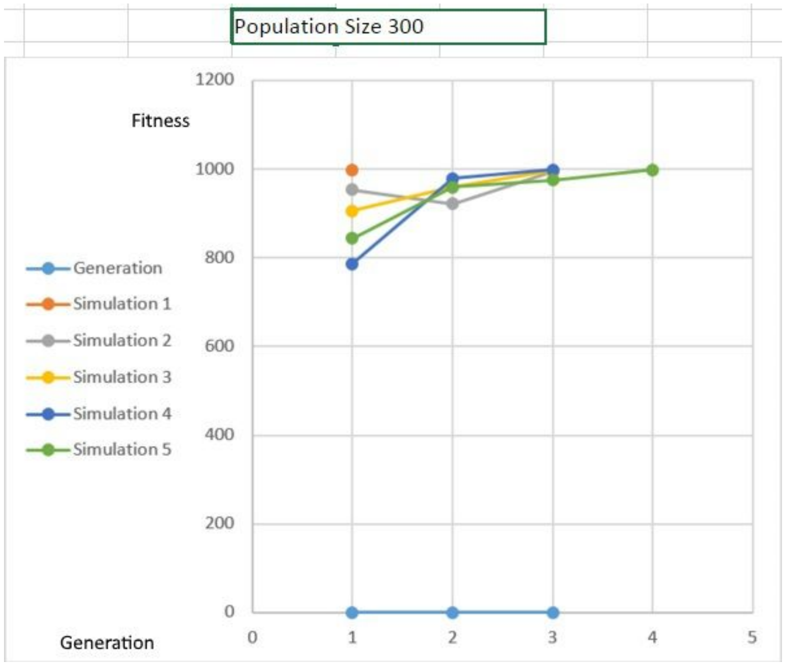
User is able to test the simulation using the shown sliders.
Mutation Rate : Range( 0 , 100 )
Population Size : Range ( 50, 500)
The fitness toggle switch can be used to alter between the fitness functions.

The fitness Effect Rate is used by fitness function 2 to determine how much of the time variant fitness effects the overall fitness.
Start and Reset button is implemented.
A debug screen is also implemented which shows fitness and moon hits for every generation.

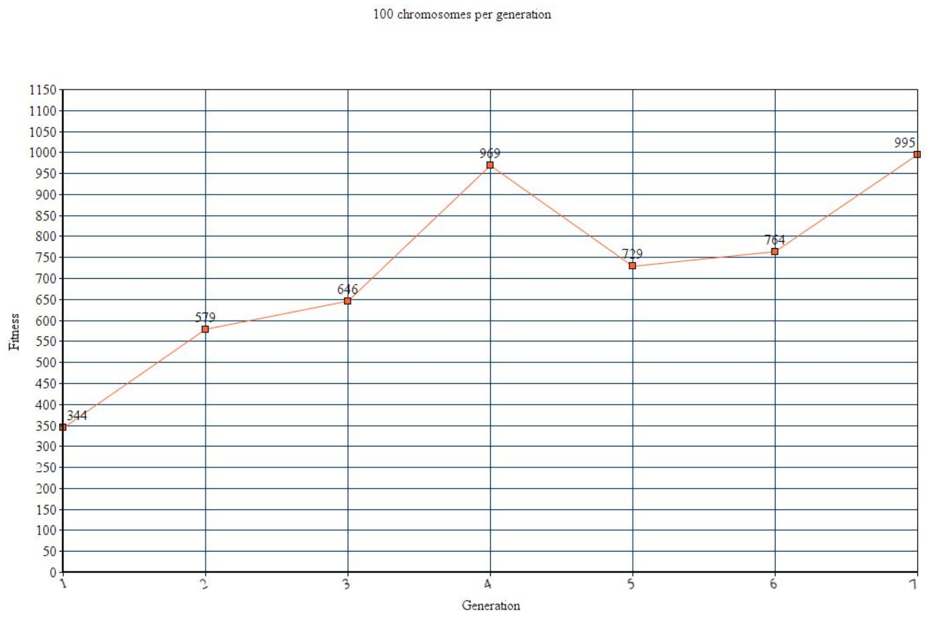
## RESULTS

### Testing with different population sizes:

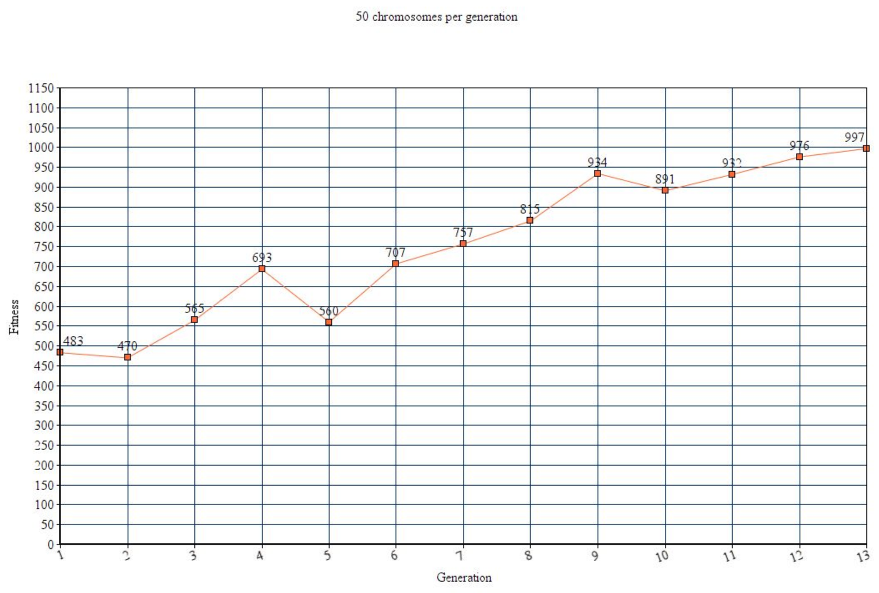




## Population Size 100

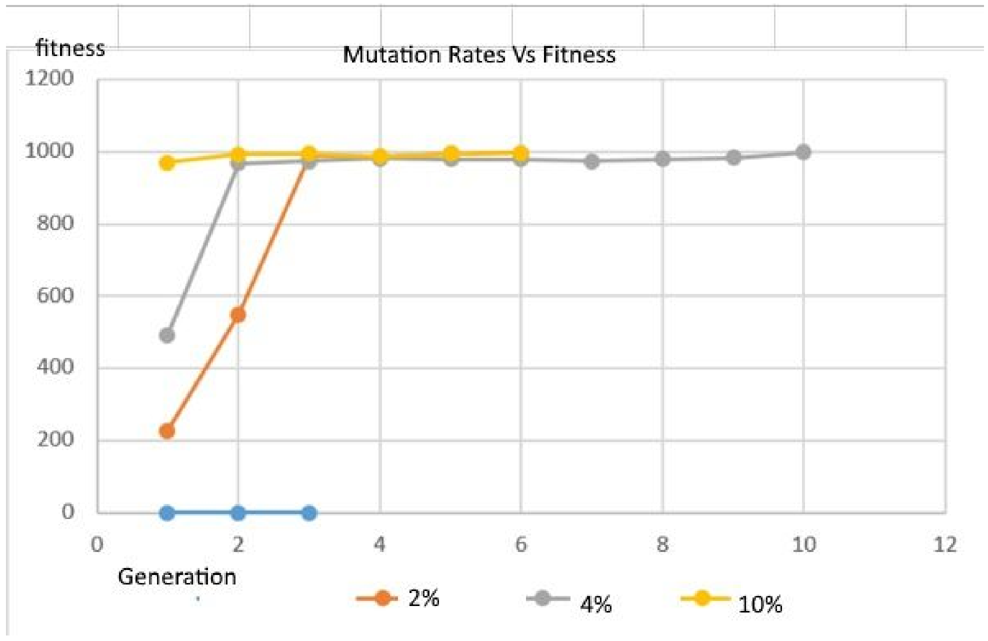


## Population Size 50



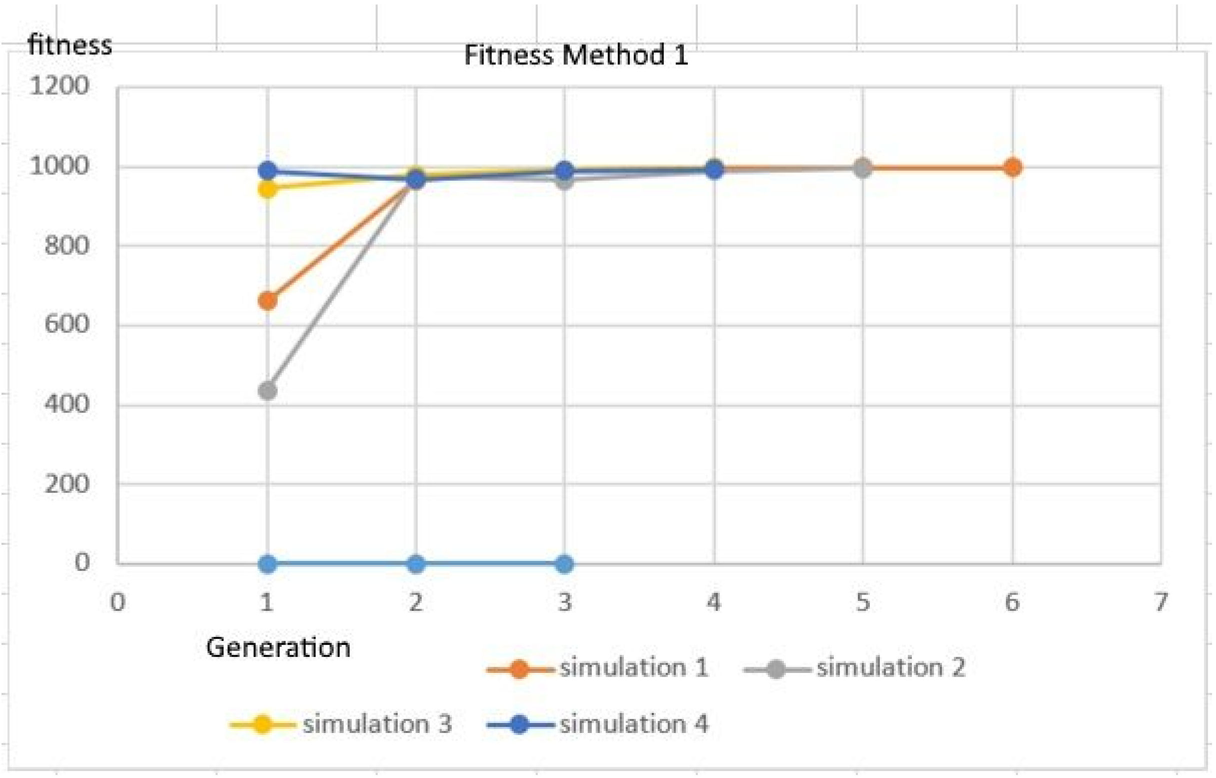
By observing the above graphs we can say that when we increase the population, the number of generations to reach the solution decrease.
We find the solution within 3 generations if the population is above 300 and lesser population is taking more number of generations to find the solution.

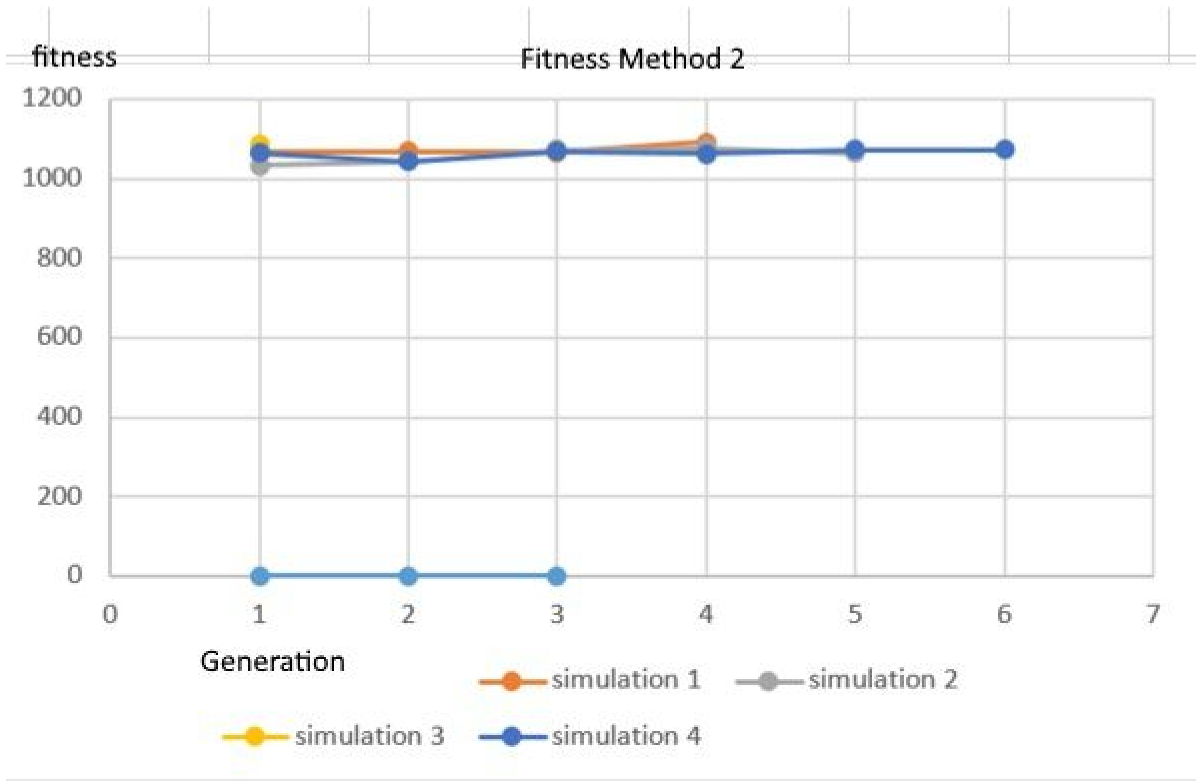
Testing with different Mutation Rates:



The Mutation rate under 10% is not much affecting the simulation as the first generation is always random and there isn't a noticeable change in the rest of the generations.

## Testing the two fitness functions





**There is not much of a noticeable change except for the average number of generations taken to complete is lesser for method 2.**

**References :**

**http://docs.unity3d.com/Manual/index.html**

**https://www.youtube.com/watch?v=wKgKSg0Nr6A**

**PART - 3**

**Effort Log**

**Vishnu Vardhan Kumar Pallati**

- **Unity Simulation Environment Development**
- **Design of GA**
- **Testing of GA & overall correctness of the project**
- **Developing GUI and Testing it**

**Nicholas Elliott**

- **Design of GA**
- **Performing Testing and getting results**
- **Supporting in completing the project in various aspects**

**Shrinivass Arunachalam balasubramanian**

- **Unity Simulation Environment development**
- **Documentation scripting**
- **Generating output results**