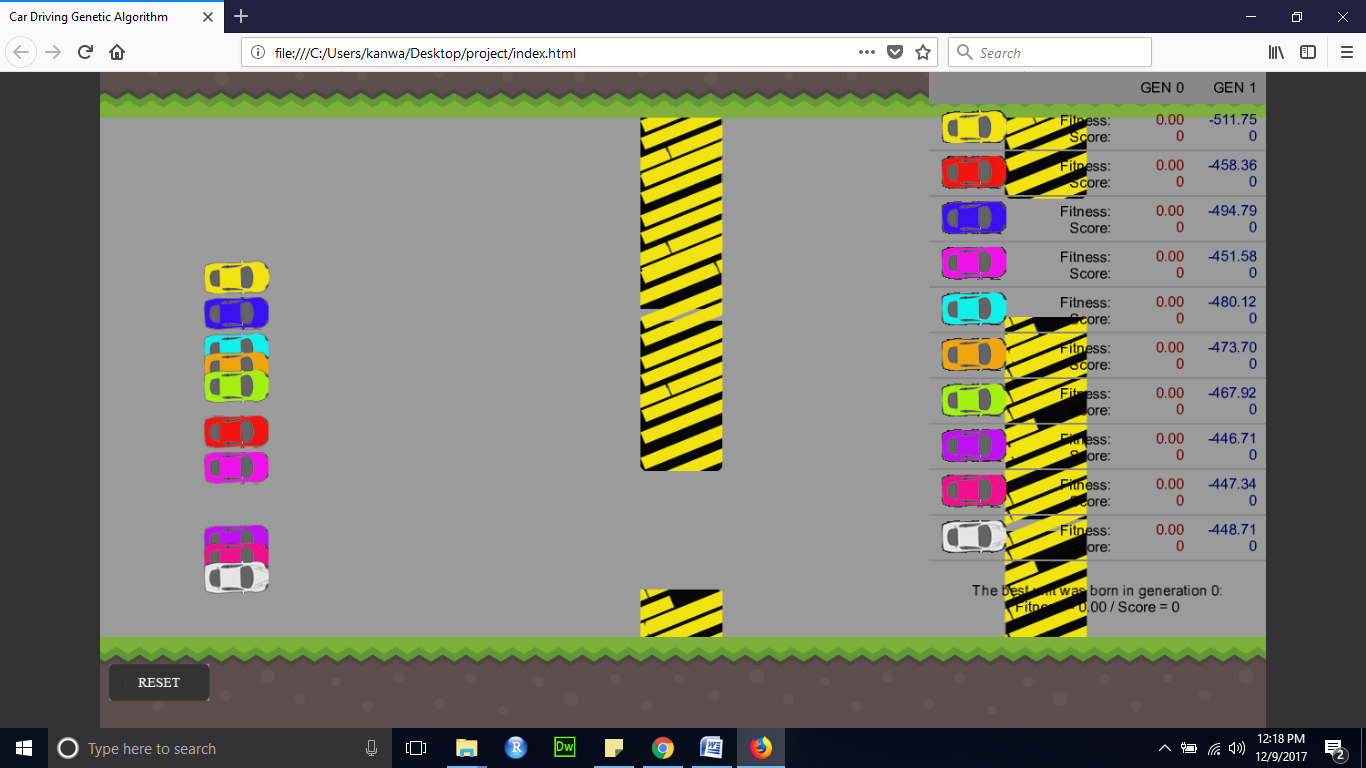
**Machine Learning for Car driving using Neural Network and Genetic Algorithm**

Here we have implemented a HTML5 project that implements a machine learning algorithm in the Car driving video game using neural networks and a genetic algorithm. The program teaches a car how to turn left and right optimally in order to cross safely through barriers for as long as possible.

All code is written in HTML5 using [Phaser framework](http://phaser.io/) for the game development and [Synaptic Neural Network library](https://synaptic.juancazala.com/) for neural network implementation.



# What Is Machine Learning Algorithm

According to Arthur Samuel in 1959, machine learning is the science of getting computers to act without being explicitly programmed. Generally speaking, this is a fine tuning process of learning that incrementally improves an initial random system.

Therefore, here is the goal to achieve an artificial intelligence which can find a proper solution from a bad system by fine tuning model parameters. To do that, machine learning algorithm uses a number of different approaches.

Specifically for this project, the main approach of machine learning algorithm (ML) is based on the NeuroEvolution (or neuro-evolution). This form of machine learning uses evolutionary algorithms such as a genetic algorithm (GA) to train artificial neural networks (ANN).

So we can say ML = GA + ANN.

# Artificial Neural Network

An artificial neural network is a subset of machine learning algorithm. It is inspired by the structure and functions of biological neural networks. These networks are made out of many neurons which send signals to each other.

Therefore, to create an artificial brain we need to simulate neurons and connect them to form a neural network.

A generic artificial neural network consists of an input layer, one or more hidden layers and an output layers. Each layer has a number of neurons. Input and output neurons are connected directly to an external environment. Finally, hidden neurons are connected between them.

In this project, each unit (car) has its own neural network used as its AI brain for playing the game. It consists of 3 layers as follows:

1. an input layer with 2 neurons representing what a car sees:

* horizontal distance to the closest gap
* height difference to the closest gap

1. a hidden layer with 6 neurons
2. an output layer with 1 neuron which provides an action as follows:

* if output > 0.5 then turn left else if output < 0.5 then turn right

# Genetic Algorithm

When we talked about machine learning algorithm, we said that a genetic algorithm is used to train and improve neural networks.

Genetic algorithm is a search-based optimization technique inspired by the process of natural selection and genetics. It uses the same combination of selection, crossover and mutation to evolve initial random population.

Here are the main steps of our genetic algorithm implementation:

* create initial population of 10 units (cars) with random neural networks
* let all units play the game simultaneously by using their own neural networks
* for each unit calculate its fitness function to measure its quality (for more details see fitness function below)
* when all units died, evaluate the current population to the next one by using genetic operators(for more details see replacement strategy below)
* go back to the step 2

# Fitness Function

In addition to the genetic algorithm (step 3), here we go with more details about fitness function – what it is and how to define it.

Since we want to evolve a population by using the best units, we need to define a fitness function.

Generally, the fitness function is the metrics to measure quality of an object. While we have a quality measure for each car, we can select the fittest units and use them to reproduce the next population.

In this project, we reward a car equally to its travelled distance. Also, we penalize it by its current distance to the closest gap. So in that way, we are making a difference between cars which travelled the same distance.

To conclude, our fitness function is the difference between the total distance covered by a car and its current distance to the closest gap.

# Replacement Strategy

In addition to the genetic algorithm (step 4), here are the steps for applying natural evolution on dying population. Basically, the best units survive and their children replace the worst units in this way:

* sort the units of the current population by their fitness ranking
* select the top 4 units (winners) and pass them directly on to the next population
* create 1 offspring as a crossover product of two best winners
* create 3 offsprings as a crossover products of two random winners
* create 2 offsprings as a direct copy of two random winners
* apply random mutations on each offspring to add some variations

**Neural Network Architecture**

To play the game, each unit (car) has its own neural network consisted of the 3 layers:

1. an input layer with 2 neurons presenting what a car sees:

* horizontal distance between the car and the closest gap
* height difference between the car and the closest gap

1. a hidden layer with 6 neurons
2. an output layer with 1 neuron used to provide an action as follows:
3. if output > 0.5 then turn left else if output < 0.5 then turn right

There is used [Synaptic Neural Network library](https://synaptic.juancazala.com/) to implement entire artificial neural network instead of making a new one from the scratch.

**The Main Concept of Machine Learning**

The main concept of machine learning implemented in this program is based on the neuro-evolution form. It uses evolutionary algorithms such as a genetic algorithm to train artificial neural networks. Here are the main steps:

1. create a new population of 10 units (cars) with a **random neural network**
2. let all units play the game simultaneously by using their own neural networks
3. for each unit calculate its **fitness** function to measure its quality as:
4. fitness = total travelled distance - distance to the closest gap
5. when all units are killed, evaluate the current population to the next one using **genetic algorithm operators** (selection, crossover and mutation) as follows:
6. 1. sort the units of the current population in decreasing order by their fitness ranking

2. select the top 4 units and mark them as the winners of the current population

3. the 4 winners are directly passed on to the next population

4. to fill the rest of the next population, create 6 offsprings as follows:

- 1 offspring is made by a crossover of two best winners

- 3 offsprings are made by a crossover of two random winners

- 2 offsprings are direct copy of two random winners

5. to add some variations, apply random mutations on each offspring.

1. go back to the step 2

**Implementation**

**Requirements**

Since the program is written in HTML5 using [Phaser framework](http://phaser.io/) and [Synaptic Neural Network library](https://synaptic.juancazala.com/) you need these files:

* **phaser.min.js**
* **synaptic.min.js**

**gameplay.js**

The entire game logic is implemented in **gameplay.js** file. It consists of the following classes:

* App.Main, the main routine with the following essential functions:
  + *preload()* to preload all assets
  + *create()* to create all objects and initialize a new genetic algorithm object
  + *update()* to run the main loop in which the car driving game is played by using AI neural networks and the population is evolved by using genetic algorithm
  + *drawStatus()* to display information of all units
* BarGroup Class, extended Phaser Group class to represent a moving barrier. This group contains a top and a bottom bar sprite.
* Bar Class, extended Phaser Sprite class to represent a Bar sprite.
* car Class, extended Phaser Sprite class to represent a car sprite.
* Text Class, extended Phaser BitmapText class used for drawing text.

**genetic.js**

The genetic algorithm is implemented in **genetic.js** file which consists of the following class:

* GeneticAlgorithm Class, the main class to handle all genetic algorithm operations. It needs two parameters: ***max\_units*** to set a total number of units in population and ***top\_units*** to set a number of top units (winners) used for evolving population. Here are its essential functions:
  + *reset()* to reset genetic algorithm parameters
  + *createPopulation()* to create a new population
  + *activateBrain()* to activate the AI neural network of an unit and get its output action according to the inputs
  + *evolvePopulation()* to evolve the population by using genetic operators (selection, crossover and mutations)
  + *selection()* to select the best units from the current population
  + *crossOver()* to perform a single point crossover between two parents
  + *mutation()* to perform random mutations on an offspring

# Conclusion

In this project we have successfully implemented AI robot for learning how to play car driving game. As a result of several iterations, we can get an almost invincible player. To achieve that goal we have used two approaches of machine learning algorithms: artificial neural networks and genetic algorithm.

As a future work, we can try to change some parameters in code and see what happens. For instance, we can change the number of neurons in hidden layer or number of units in population. Also, we can try to change the fitness function somehow. Furthermore, change some physical parameters such as the distance between barriers and so on!

Also, we can try to apply the same idea of evolution to some other games!

**Teammate:**

**Fei Feng fxf160330 Uday Singh uxs160630**