PA026 – Training self-driving cars using genetic algorithm

Peter Hutta

Spring 2019

1 Introduction

The goal of this project was to create a program which would train agents/cars to drive using genetic algorithm (GA). It would be implemented in Unity 3D game engine. Users would be able to interact with the training process, change hyper-parameters of underlying algorithm and properties of agents.

There are several working implementations of this topic [1, 2, 3]. However, all of them use neural network with static structure and GA is used to train its weights. My implementation used algorithm called NeuroEvolution of Augmenting Topologies (NEAT), which evolves weights and topology at the same time. This project also uses multiple training and testing tracks.

2 NEAT algorithm

NEAT is a method for evolving artificial neural networks using GE [4]. It evolves both weighting parameters and structure of networks, trying to find balance between fitness of the solution and their diversity.

Networks in the initial population are as simple as possible, often having no connections at all (input and output neurons are not connected). During evolution, algorithm is adding new neurons and connections between existing ones. This leads to having small resulting networks.

In ordinary GE it can happen that two individuals encode the same behavior but with different genotype. When subjected to crossover, their offspring is likely to be worse than their parents. This phenomenon is called competing conventions. In NEAT this is solved by keeping historical markings of new structures. It is assigned unique number and all mutations containing such structure are assigned the same number. During crossover genes with matching numbers are kept and differing genes are exchanged.

NEAT also works with concept of species. It is a subdivision of population into multiple groups of individuals, which is based on the similarity of individuals.

Probability of crossover within one specie is much higher than between different species. Promotion of mating of similar parents causes, that children are less likely to be worse than their parents.

3 Implementation

Project was implemented in C# in Unity 2018.3.2f1. I have chosen SharpNEAT as a complete implementation of NEAT written in C# / .NET created by Colin Green [5].

Each agent has attached CarController component to it, which acts as its brain and contains underlying neural network. During each update, ray is cast in five different directions (front, front left, left, front right, right). If the ray hits a wall a value is computed using following formula:

$$input = 1 - \frac{wall\_distance}{sensor\_range}$$

These five values are then used as an input to the neural network. Output layer consists of two nodes and both are transformed to the range <-1,1>. First one together with rotation speed indicates rotation angle. Second one is used as an acceleration.

Fitness of each individual is computed in multiple steps. It takes cars position, percentage of traveled road pieces and time spend on the right side into consideration. At first, its position each lap is used in exponential decay formula, penalizing slow cars:

$$fitness_{position} = \sum_{i}^{laps} 100 * e^{-0.02*(position_i - 1)}$$

Then it is computed right side ratio, which takes number of frames spend on sides of the road and penalizes left side:

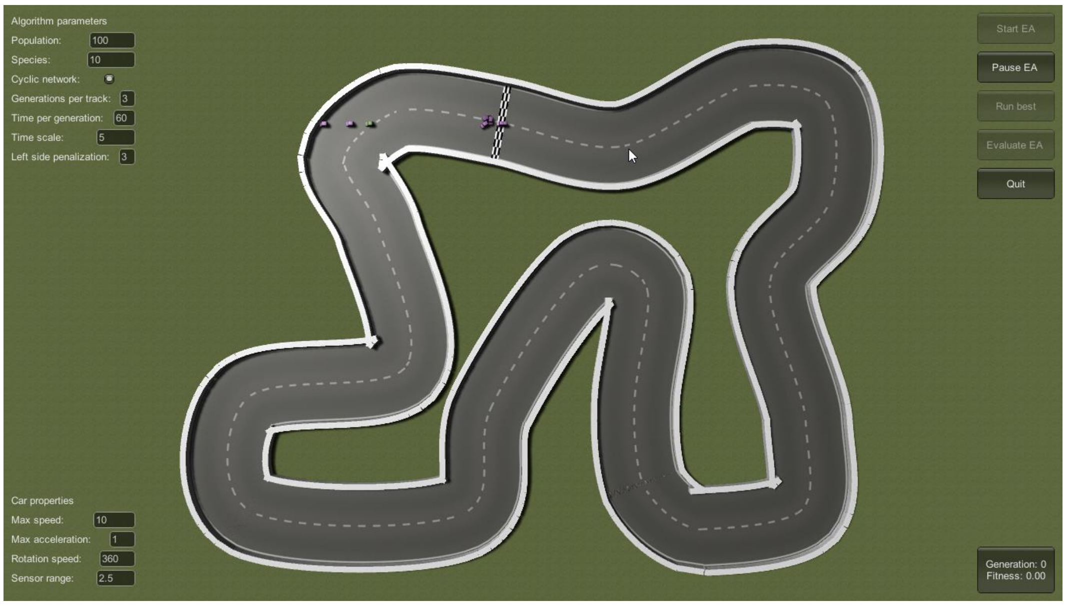
$$ratio_{right} = \frac{frames_{right}}{frames_{right} + penalization * frames_{left}}$$

Resulting fitness is computed using these values and percentage of traveled pieces during last lap:

$$fitness = \left(fitness_{position} + \frac{100 * current_{piece}}{track\_length}\right) * \max(ratio_{right}, 0.2)$$

After car hits a wall it is no longer allowed to drive and it is stopped.

Users can interact with the learning process. They can set multiple algorithm parameters (population size, specie count, generations per track …) and car properties (max speed/acceleration, sensor range …). It is also possible to pause/restart computation, run evaluation and run best individual. Each species of cars has assigned unique color to it for graphical recognition.



Algorithm parameters
Population: 100
Species: 10
Cyclic network:
Generations per track: 3
Time per generation: 60
Time scale: 5
Left side penalization: 3

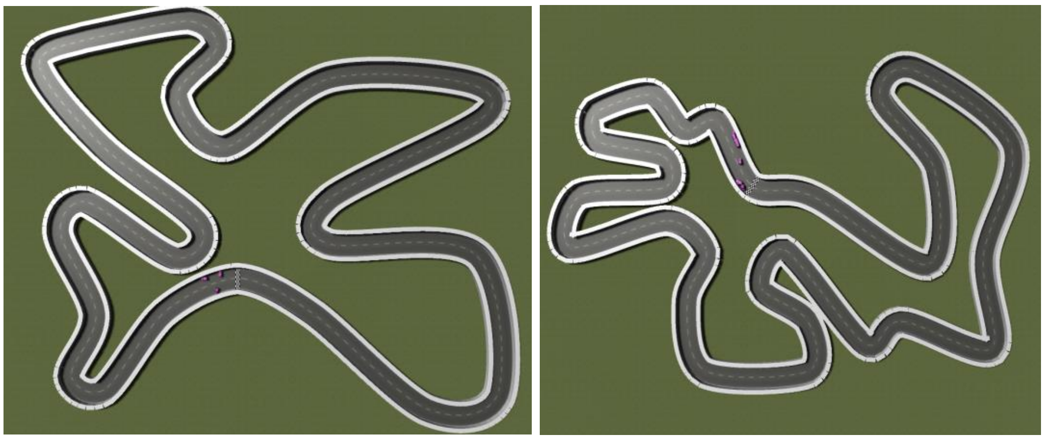
Car properties
Max speed: 10
Max acceleration: 1
Rotation speed: 360
Sensor range: 2.5

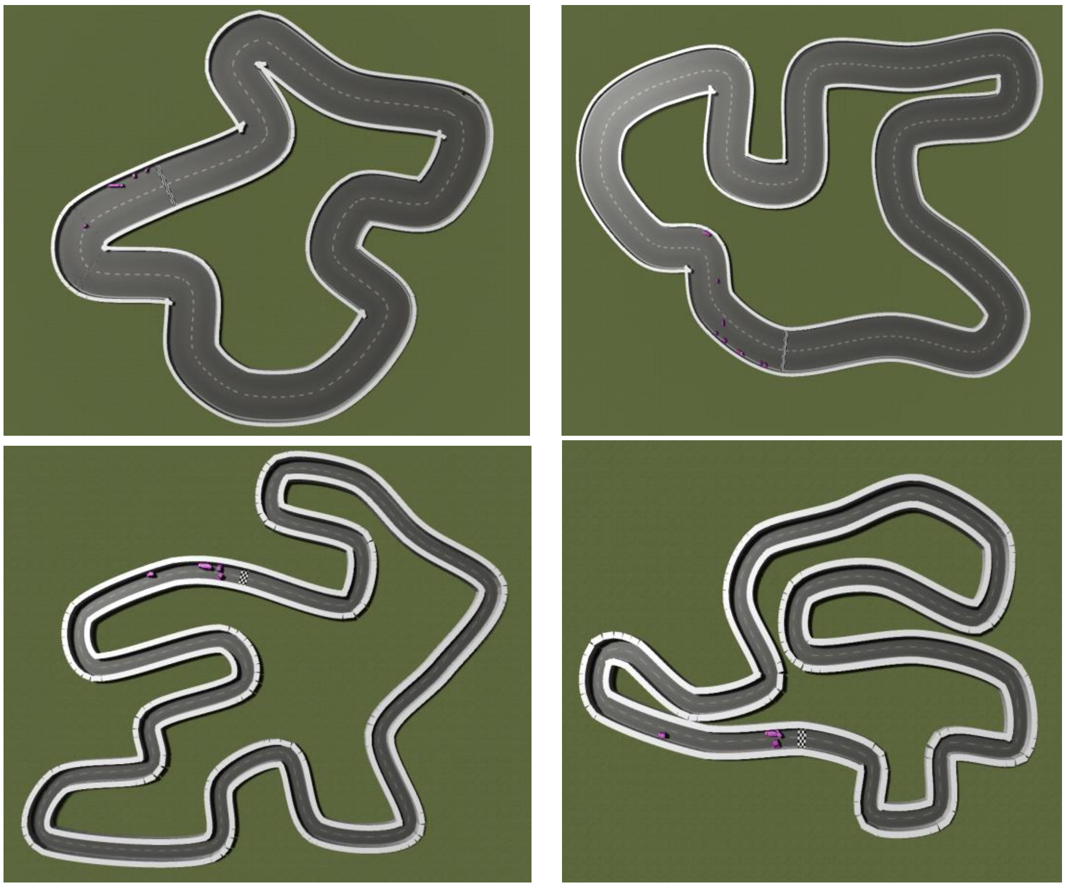
Generation: 0
Fitness: 0.00

4 Evaluation and testing

During training, it is used 11 unique tracks of differing width and shape. After each pass, their order is randomized. Users can set number of generations and seconds spend on each track.

There are 6 testing tracks. They are combination of narrow/normal/wide and clockwise/counterclockwise shapes.



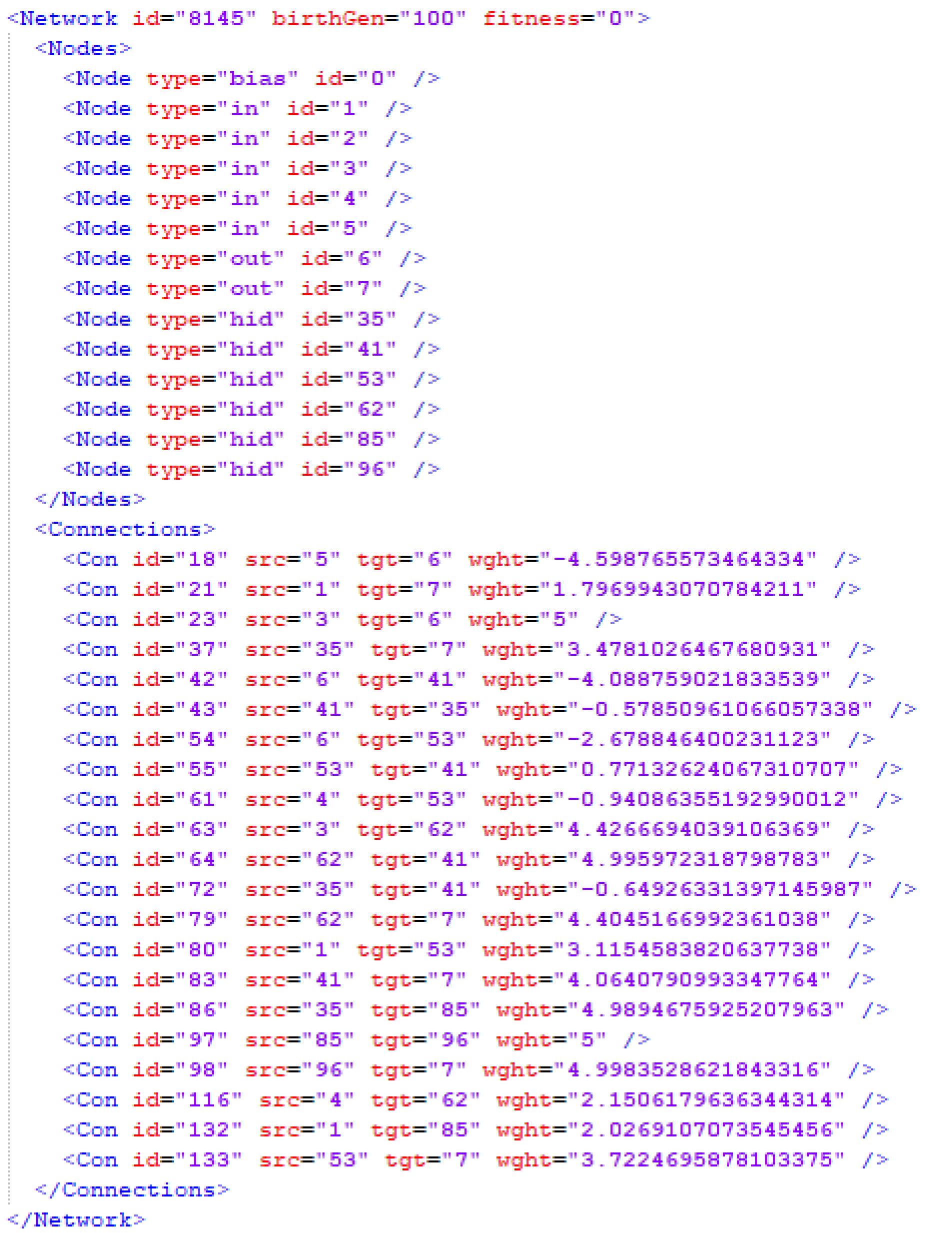


After each testing track, results are saved in csv file with various statistical data. Also, whole generation is ranked and after evaluation the individual with best average rank is outputted to file *Build\Car Experiment_Data\best_gen#.xml*.

5 Results

Training process was carried out each time with different parameters and it ran over several hundred generations. The latest generation underwent testing after each hundred generation. Then it was average ranked by individual's fitness gained on related testing track. First one was outputted as the best individual out of the whole generation.

The best configuration found so far is already preselected. After first evaluation there was an individual with average rank 1 (rounded down) and it didn't crash on neither testing track. Its gene is encoded in file *car.champ.xml* and can be viewed on the picture below.



```xml
<Network id="8145" birthGen="100" fitness="0">
  <Nodes>
    <Node type="bias" id="0" />
    <Node type="in" id="1" />
    <Node type="in" id="2" />
    <Node type="in" id="3" />
    <Node type="in" id="4" />
    <Node type="in" id="5" />
    <Node type="out" id="6" />
    <Node type="out" id="7" />
    <Node type="hid" id="35" />
    <Node type="hid" id="41" />
    <Node type="hid" id="53" />
    <Node type="hid" id="62" />
    <Node type="hid" id="85" />
    <Node type="hid" id="96" />
  </Nodes>
  <Connections>
    <Con id="18" src="5" tgt="6" wght="-4.598765573464334" />
    <Con id="21" src="1" tgt="7" wght="1.7969943070784211" />
    <Con id="23" src="3" tgt="6" wght="5" />
    <Con id="37" src="35" tgt="7" wght="3.4781026467680931" />
    <Con id="42" src="6" tgt="41" wght="-4.088759021833539" />
    <Con id="43" src="41" tgt="35" wght="-0.57850961066057338" />
    <Con id="54" src="6" tgt="53" wght="-2.678846400231123" />
    <Con id="55" src="53" tgt="41" wght="0.77132624067310707" />
    <Con id="61" src="4" tgt="53" wght="-0.94086355192990012" />
    <Con id="63" src="3" tgt="62" wght="4.4266694039106369" />
    <Con id="64" src="62" tgt="41" wght="4.995972318798783" />
    <Con id="72" src="35" tgt="41" wght="-0.64926331397145987" />
    <Con id="79" src="62" tgt="7" wght="4.4045166992361038" />
    <Con id="80" src="1" tgt="53" wght="3.1154583820637738" />
    <Con id="83" src="41" tgt="7" wght="4.0640790993347764" />
    <Con id="86" src="35" tgt="85" wght="4.9894675925207963" />
    <Con id="97" src="85" tgt="96" wght="5" />
    <Con id="98" src="96" tgt="7" wght="4.9983528621843316" />
    <Con id="116" src="4" tgt="62" wght="2.1506179636344314" />
    <Con id="132" src="1" tgt="85" wght="2.0269107073545456" />
    <Con id="133" src="53" tgt="7" wght="3.7224695878103375" />
  </Connections>
</Network>
```

Also, I have found out that training is worth running only for 100-200 generations. After that average rank is increases, fitness decreases and more cars crash to wall.

One of the possible ways how to improve this project is to use version of NEAT called HyperNEAT. It is used to evolve large scaled neural network, which would allow the use of convolutional networks and Synthia dataset. Synthia is a collection of photo-realistic frames rendered from virtual city coming with sematic annotations. It has variety of dynamic objects, multiple seasons and different lighting and weather conditions.

6 Installation

Release build for Windows (64-bit) can be launched from *Build\Car Experiment.exe.* Alternatively, unity project in *pa026-project* can be opened using Unity 2018.3.2f1. After opening scene *pa026-project\Assets\CarExperiment\Scenes\Scene1.unity*. it is possible to launch development build.

You can copy configuration file of best individual (*car.champ.xml*) to Unitys persistent data path [6] (*%userprofile%\AppData\LocalLow\DefaultCompany\Car Experiment* on Windows) to run it for yourself.

7 References

1. https://ashwinvaidya.com/blog/self-driving-car-using-genetic-algorithm-in-unity/
2. https://github.com/ArztSamuel/Applying_EANNs
3. https://tutorials.retopall.com/index.php/2019/03/21/self-driving-cars-3d-simulation/
4. https://www.cs.ucf.edu/~kstanley/neat.html
5. https://sharpneat.sourceforge.io/
6. https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html