ABSTRACT

Developing a game bot opens the path of implementing the problem-solving ideas in real life situations. Here, the goal of our project is to create a bot for a game environment having a locomotive that is able to adapt to changing terrains.

Recent advances in Deep Reinforcement learning and genetic algorithms have allowed autonomous agents to perform well in 2D and some 3D games using only raw pixels to make decisions, this offers a promising methodology for developing an algorithm to be able to maneuver changing terrains inside of a game environment.

INTRODUCTION

In Reinforcement Learning we pass a reward, positive or negative depending on the action the system took, and the algorithm needs to learn what actions can maximize the reward, and which need to be avoided.

To use reinforcement learning successfully in situations approaching real-world complexity, agents are confronted with a difficult task, they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations.

Thus, this thinking leads us to Deep Reinforcement Learning which uses a deep neural network to approximate the values.

While working on the project, we found out that using reinforcement learning was slow and insufficient. Although this approach is not wrong at all, this is more suited for small environments and quickly loses its feasibility when the number of states and actions in the environment increases. It is slow when dealing with a terrain that is randomly generated and thus, requires a lot of training.

Therefore, we have used the NEAT(NeuroEvolution of Augmenting Topologies) algorithm, which is a genetic algorithm for creating our model. This approach significantly reduces the training time as genetic algorithms like NEAT are very successful for dealing with such types of gameplay.

LITERATURE SURVEY

1) Human-level control through reinforcement learning

Reference : https://www.nature.com/articles/nature14236?wm=book_wap_0005

The theory of reinforcement learning provides a normative account , deeply rooted in psychological and neuroscientific perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems.

While reinforcement learning agents have achieved some successes in a variety of domains, their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

2) Playing FPS Games with Deep Reinforcement Learning

Reference :https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14456

First-Person-Shooting (FPS) game in a 3D environment involves a wide variety of skills, such as navigating through a map, collecting items, recognizing and fighting enemies, etc.

This project demonstrates playing deathmatches in Terrain adaptive locomotive games using only the pixels on the screen.The problem is divided into two phases: navigation (exploring the map to collect items and find obstructions) and action (avoiding obstructions when they are observed), and uses separate networks for each phase of the game.
In FPS they have used the algorithm to maximize the game score, here we would

implement a similar algorithm to maximize the distance travelled by the car.

3) Playing Atari with Reinforcement Learning

Reference : https://arxiv.org/abs/1312.5602)

The goal was to create a single neural network agent that was able to successfully learn to play as many of the games as possible. The network was not provided with any game-specific information or hand-designed visual features, and was not privy to the internal state of the emulator. It learned from nothing but the video input, the reward and terminal signals, and the set of possible actions—just as a human player would. Furthermore the network architecture and all hyperparameters used for training were kept constant across the games.

This network outperformed all previous RL algorithms on six of the seven games they attempted and surpassed an expert human player on three of them.

4) Deep Reinforcement Learning from Self-Play in Imperfect-Information Games

Reference : https://arxiv.org/abs/1603.01121

Games have a tradition of encouraging advances in artificial intelligence and machine learning. One motivation for studying recreational games is to develop algorithms that will scale to more complex, real-world games such as airport and network security, financial and energy trading, traffic control and routing

An optimal solution to these games would be a Nash equilibrium, i.e. a strategy from which no agent would choose to deviate. While many machine learning methods have achieved near-optimal solutions to classical, perfect-information games, these methods fail to converge in imperfect-information games.

This paper tries to introduce the first scalable end-to-end approach to learning approximate Nash equilibria without prior domain knowledge.

5)Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

Reference : https://arxiv.org/abs/1612.00380

The combination of modern Reinforcement Learning and Deep Learning approaches holds the promise of making significant progress on challenging applications requiring both rich perception and policy-selection.

The Arcade Learning Environment provides a set of Atari games that represent a useful benchmark set of such applications.

Planning-based approaches achieve far higher scores than the best model-free approaches, but they exploit information that is not available to human players, and they are orders of magnitude slower than needed for real-time play. The main goal in this work is to build a better real-time Atari game playing agent than DQN. The central idea is to use the slow planning-based agents to provide training data for a deep-learning architecture capable of real-time play. This paper proposed new agents based on this idea and show that they outperform DQN.

6) Deep Reinforcement Learning framework for Autonomous Driving

Ref. : https://www.ingentaconnect.com/content/ist/ei/2017/00002017/00000019/art00012

Reinforcement learning is considered to be a strong AI paradigm which can be used to teach machines through interaction with the environment and learning from their mistakes. Despite its perceived utility, it has not yet been successfully applied in automotive applications

Motivated by the successful demonstrations of learning of Atari games and Go by Google DeepMind, we propose a framework for autonomous driving using deep reinforcement learning. This is of particular relevance as it is difficult to pose autonomous driving as a supervised learning problem due to strong interactions with the environment including other vehicles, pedestrians and roadworks

7) Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning

Reference : https://dl.acm.org/doi/10.1145/2897824.2925881

Humans and animals move with grace and agility through their environment. In animation, their movement is most often created with the help of a skilled animator or motion capture data. The use of reinforcement learning (RL) together with physics-based simulations offers the enticing prospect of developing classes of motion skills from first principles.

This requires viewing the problem through the lens of a sequential decision problem involving states, actions, rewards, and a control policy.

Given the current situation of the character, as captured by the state, the control policy decides on the best action to take, and this then results in a subsequent state, as well as a reward that reflects the desirability of the observed state transition.

8) Augmenting the NEAT algorithm to improve its temporal processing capabilities

Reference: https://ieeexplore.ieee.org/abstract/document/6889488

This paper is concerned with the incorporation of new time processing capacities to the Neuroevolution of Augmenting Topologies (NEAT) algorithm. This algorithm is quite popular within the robotics community for the production of trained neural networks without having to determine a priori their size and topology. However, and even though the algorithm can address temporal processing issues through its capacity of establishing feedback synaptic connections, that is, through recurrences, there are still instances where more precise time processing may go beyond its limits. In order to address these cases, in this paper we describe a new implementation of the NEAT algorithm where trainable synaptic time delays are incorporated into its toolbox. This approach is shown to improve the behavior of neural networks obtained using NEAT in many instances. Here, we provide some of these results using a series of typical complex time processing tasks related to chaotic time series modeling and consider an example of the integration of this new approach within a robotic cognitive architecture.

9) Automatic Task Decomposition for the NeuroEvolution of Augmenting Topologies (NEAT) Algorithm

Reference: https://link.springer.com/chapter/10.1007/978-3-642-29066-4_1

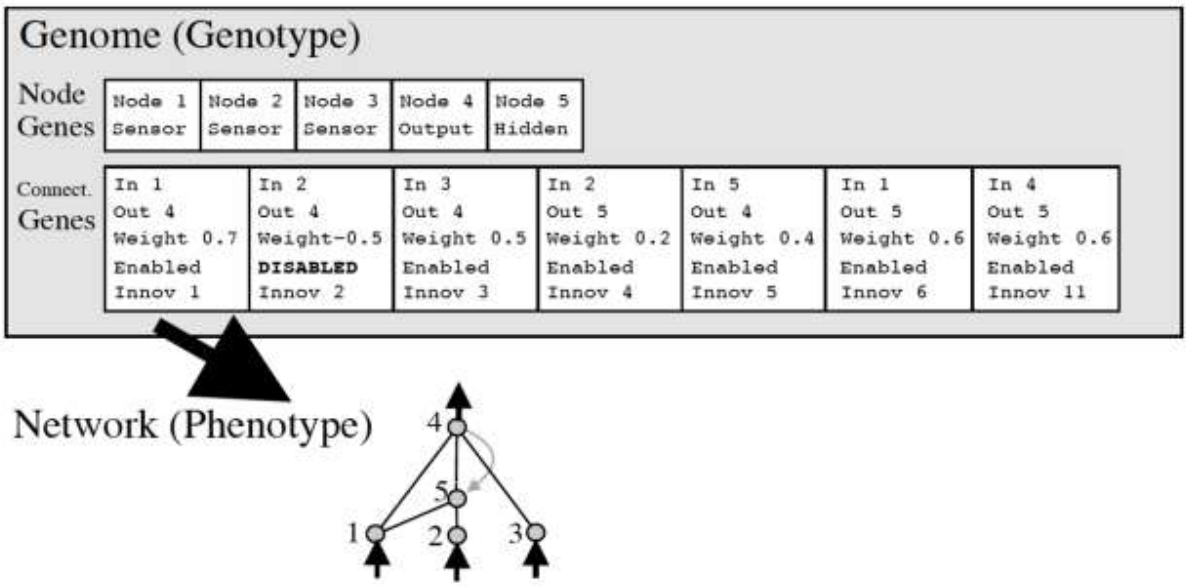
Neuroevolution, the process of creating artificial neural networks through simulated evolution, can become impractical for arbitrarily complex problems

requiring large or intricate neural network architectures. The modular feed forward neural network (MFFN) architecture decomposes a problem among a number of independent task specific neural networks, and is suggested here as a means of managing neuroevolution for complex problems. We present an algorithm for evolving MFFN architectures based on the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. The algorithm proposed here, denoted MFF-NEAT, outlines an approach to automatically evolving, attributing fitness values and combining the task specific networks in a principled manner.

Proposed Methodology

In our project, we are using **NEAT**(NeuroEvolution of Augmenting Topologies) algorithm and just like with all evolutionary algorithms, a population of ANNs is intrinsically and easily processed in parallel—if you have 100 ANNs in the population and 100 processors, you can evaluate all of those networks at the same time, in the time it takes to evaluate a single network.
NEAT uses Speciation which simply splits up the population into several species based on the similarity of topology and connections. NEAT uses historical markings in its encoding, this becomes much easier to measure. The important part to note is that individuals in a population only have to compete with other individuals within that species. This allows for new structure to be created and optimized without fear that it will be eliminated before it can be truly explored. More than that, NEAT takes things one step forward through something called explicit fitness sharing. That means that individuals share how well they are doing across the species, boosting up higher performing species, though still allowing for other species to explore their structure optimization before being out evolved.

The NEAT algorithm chooses a direct encoding methodology. It simply has two lists of genes, a series of nodes and a series of connections.

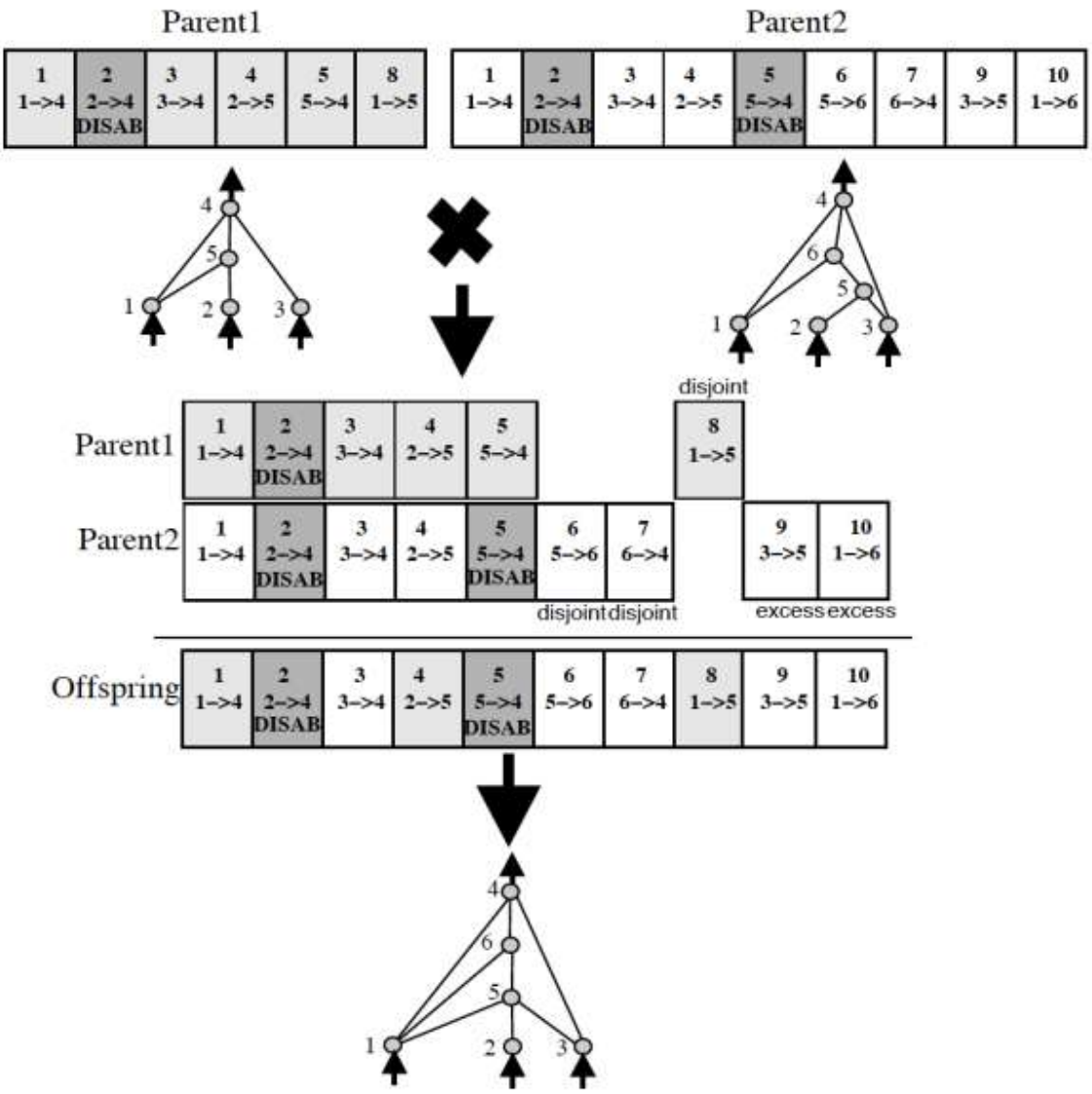


Input and output nodes are not evolved in the node gene list. Hidden nodes can be added or removed. As for connection nodes, they specify where a connection comes into and out of, the weight of such connection, whether or not the connection is enabled, and an innovation number.

Competing Convention

Another big issue with evolving the topologies of neural networks is something that the NEAT paper calls “competing conventions.” The idea is that just blindly crossing over the genomes of two neural networks could result in networks that are horribly mutated and non-functional. If two networks are dependent on central nodes that both get recombined out of the network, we have an issue.

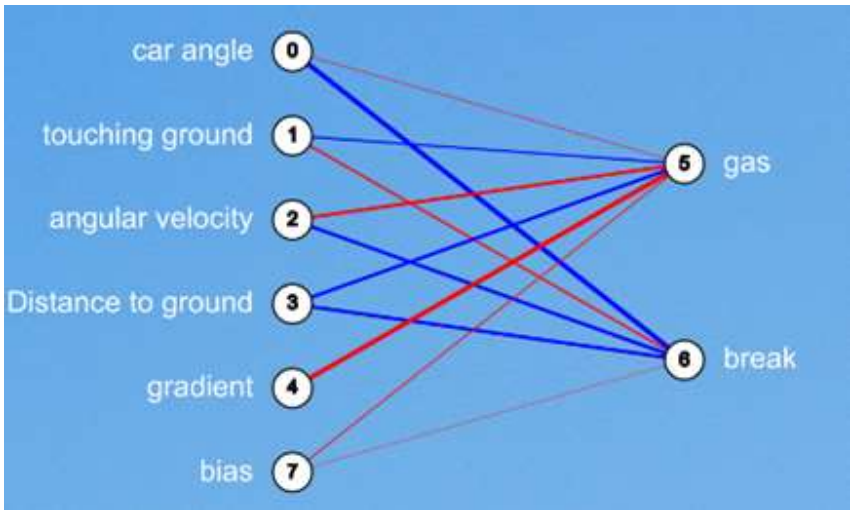
NEAT tackles this issue through the usage of historical markings (as seen above). By marking new evolutions with a historical number, when it comes time to crossover two individuals, this can be done with much less chance of creating individuals that are non-functional. Each gene can be aligned and (potentially) crossed-over. Each time a new node or new type of connection occurs, a historical marking is assigned, allowing easy alignment when it comes to breeding two of our individuals.



Implementation

Our neural net contains 6 inputs and 2 outputs. In order to use NEAT algorithm on our box2D game, we will be using NEATjs, which is a template for NEAT algorithm to work with JavaScript based games like ours. As our game uses

Box2D, it has inbuilt methods to output certain physics values, that we will be feeding to our neural network.



The NEAT based player bot has 6 inputs:

- Car angle: The angle that the centre of car makes with the horizontal at any given time
- Touching ground : The car has back and front wheels, and we return a boolean true or false value for each of the wheels based on whether they are touching the ground at any given time.
- Angular velocity: When the car is accelerated, a torque is generated, that can introduce an angular motion. This parameter is also fed to the network.
- Distance to ground: The vertical distance between the ground and the car(can change depending on position of wheels over the terrain)
- Gradient: The gradient of the terrain
- Bias: Network input bias, adjusted with each round.

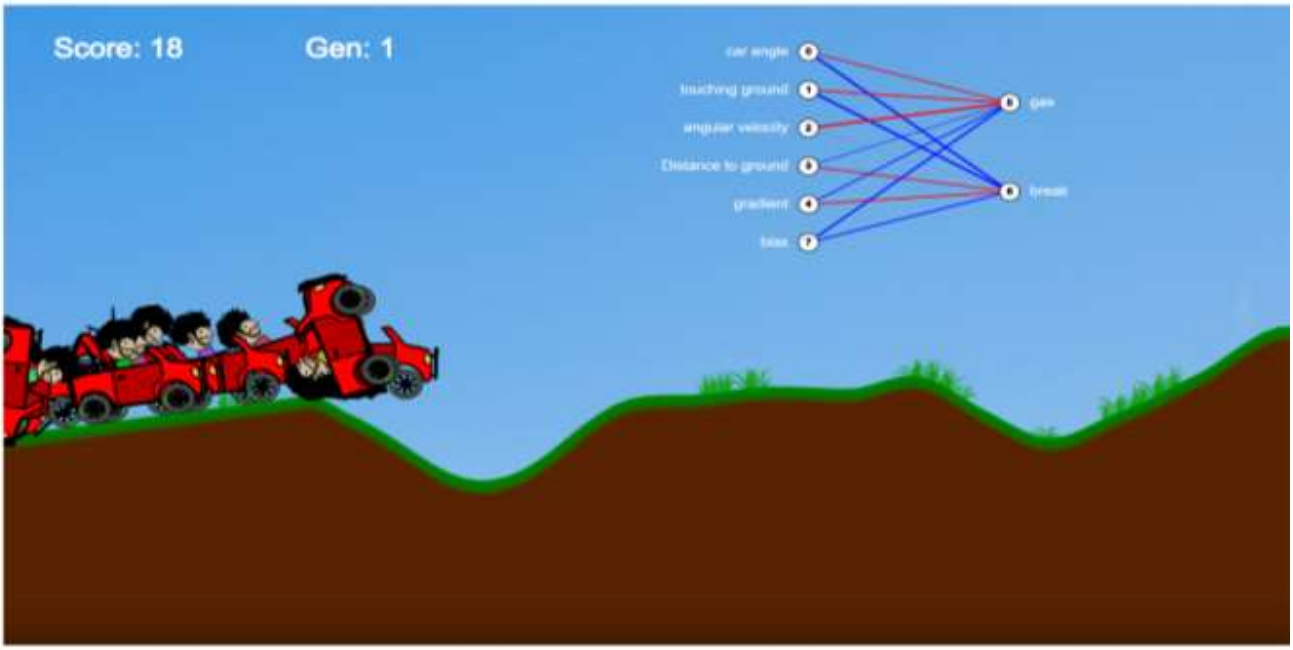
Using our genetic algorithm, we use a population size of 750 Players for each generation. A large population size is necessary to train the model properly(a critical mass is needed, otherwise the learning is slowed down significantly or is outright unsuccessful). However, having 750 players running at a single instance is

very resource intensive and can crash the browser running the game. Therefore, we use a batch learning process, where we divide the population of each generation into batches of 50 players. This way, only 50 Player processes are spawned at a given moment, but the population size for the single generation still remains the same.

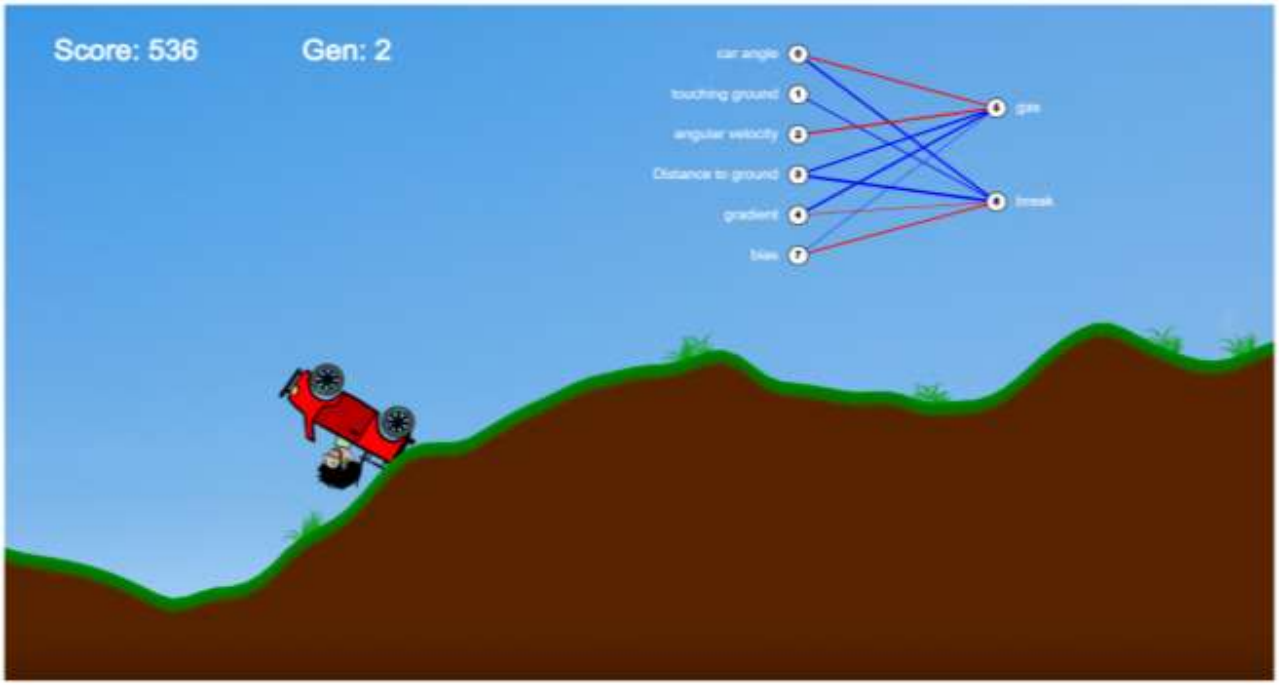
Each player in the given population, has a fitness score, based on how far it is able to go over the terrain. This fitness helps decide the winners for one generation. The players with the highest fitness are then selected and mutated, and their genes are passed onto the next generation. This progressively helps create better players that have the genes of the previous winning players and this approach helps significantly reduce our training time.

Result

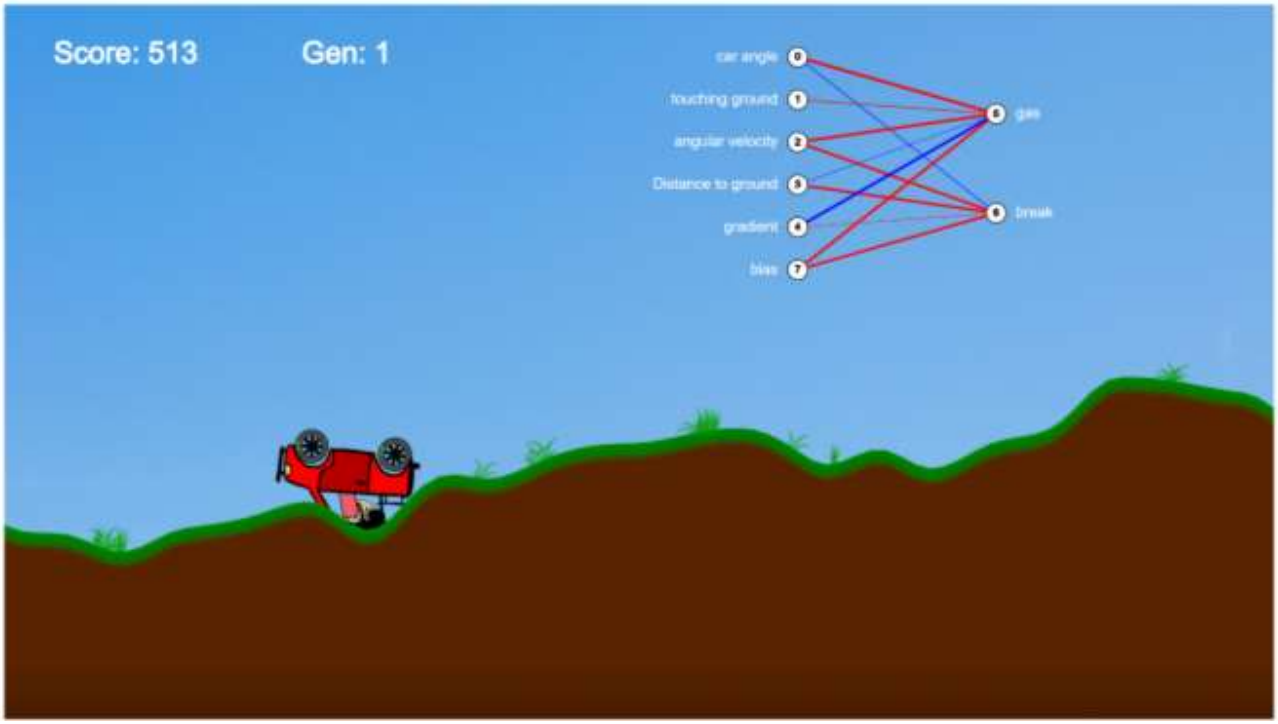
TerrainBOT



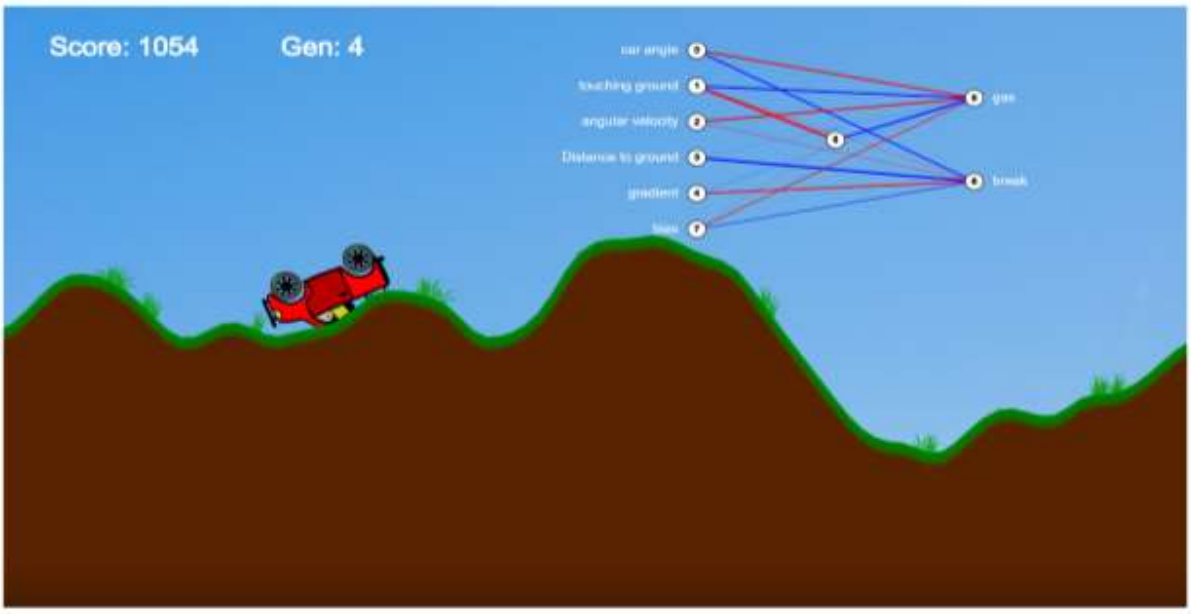
TerrainBOT

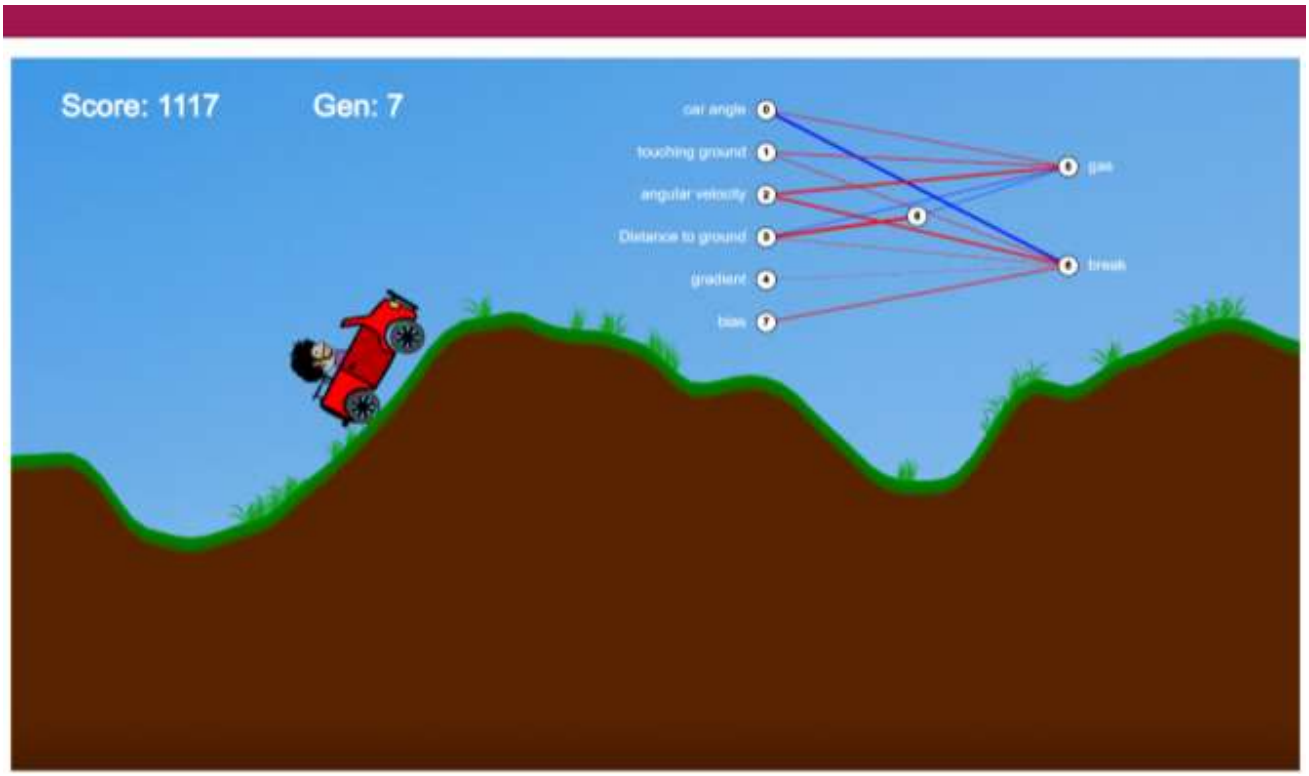


TerrainBOT



TerrainBOT





Conclusion:

Using the NEAT algorithm along with a population size of 750 Players and using a batch learning approach of 50 Players per batch, the model is able to perform remarkably well after just 2 to 3 generations and it is able to play well beyond general human limits after reaching generation 7. This is in contrast to other algorithms such as deep reinforcement learning, which require a much longer training time. Therefore, we can conclude that using genetic algorithms like NEAT for playing such types of games seems like a suitable approach and we can use them to create gameplay bots for similar genres of games.