Simulation and Comparison of Reinforcement Learning Algorithms

Konstantinos Spyropoulos
Department of Informatics
University of Piraeus
Piraeus, Greece
kostas.spy93@gmail.com

Dr. Dionysios N. Sotiropoulos, PhD
Department of Informatics
University of Piraeus
Piraeus, Greece
dsotirop@unipi.gr

***Abstract*—The field of Artificial intelligence in modern era is an important tool for the implementation of modern ideas and the development of many useful applications. Machine learning provides a variety of techniques and algorithms for problem solving where classical programming is limited. A similar problem is presented in this work, which is a theoretical and practical approach in the field of machine learning aiming in finding suitable techniques for solving complex problems. Creating intelligent artificial agents to solve human-related problems is a major challenge for the artificial intelligence community. Particularly important is the understanding of the dynamic environment in which they operate and their interaction with it, just like humans do in the physical world. The field that specializes in creating such agents is called Reinforcement Learning. With the term Reinforcement Learning, we refer to the methods through which a system of algorithms 'learns' to interact within a structured environment after trial and error. This learning is done by exploring the environment through actions and rewards given by the environment to achieve a goal with optimal effort. This research was developed on the idea of the simple chase game, which aims to create an environment and the agents that play in it. In this game, hunting agents are tasked with catching the opponents, while the hunted agents try to stay free. The goal of the work is, apart from understanding the rules and getting the right decisions to achieve victory, finding new strategies that will lead to optimal results and comparing the most valuable algorithms. The proposed approach has been chosen after analysing a wide range of Reinforcement Learning algorithms in a graphical emulation environment. The results achieved show the best implementation technique, but at the same time prospective improvements are also highlighted in the way of getting the right decisions but also in dealing with more complex environments and rules of the game.**

***Index Terms*—Reinforcement Learning, Q Learning, Deep Q Learning, NEAT, Learning Algorithms**

I. Introduction

Learning in nature comes from interaction of living organisms with their environment. Due to this interaction, living organisms evolve, they learn from their actions and the resulting consequences and adapt in order to achieve certain goals that are extremely important to their survival. Reinforcement learning is based in this core idea and constitutes a major part in the machine learning field that faces decision making problems by gaining experience via environment interactions.

Specifically, inside a system, an agent takes actions and gains rewards which evaluate that action, depending on how much this action contribute to achieving a certain goal. The agent's final goal is to select the right decisions to maximize the aggregated reward. [1]

What separates reinforcement learning methods from the other machine learning methods (e.g., supervised learning) is that in RL the system provides only partial feedback to the agent from its predictions. Simultaneously, those predictions can have major influence in long term future states of the agent. As a result, time plays an important role in agent's learning, where every decision directly and indirectly affects every possible state of the system until the completion of the goal.

Another important parameter of RL is that the training exists through trial and error. The agent does not have the full control of the environment which it exists, so it has to collect information which will determine its behaviour. It is important to also consider that using this method, it results to the exploration/exploitation dilemma which will be further analysed in the following sections.

In this current study, a complete agent learning system was developed in a graphical environment that consists of certain rules. The main goal of the development is the usage of different techniques and algorithms and their comparison to find the one the fits in this specific problem. Simultaneously, throughout the experiments, certain conclusions were derived that describes the way each method helps in specific occasions of the problem. Equally important is the comparison of the decisions and actions of the agents with the human perspective and the different logic they follow in cases like the referred problem. [2]–[4]

This and other similar experiments, which can be resolved using reinforcement learning, can be implemented in real world problems. Examples like these can be found anywhere in the real world, especially with the uprise of computational power of computer systems.

II. Methodology

A. Architecture

The main structure of a Reinforcement Learning system consists of two parts, the agent, and the environment. The environment defines the area where the agent takes actions, providing him the state for every time t. The agent, where the RL algorithm is implemented, decides based on the current state. In the following moment t, the environment provides to the agent the new state and its reward. Following up, the agent reviews and evaluates his actions using that reward and decides again his next actions. This loop continues until the environment sends a terminal state, where the episode ends. We define episode as the above repetition until the terminal state. Terminal state can be send when the agent fulfils his purpose or when he surpasses a movement threshold in the environment.

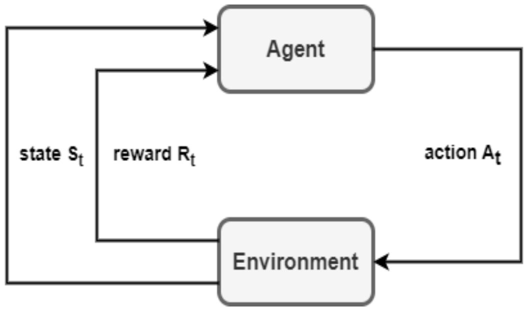

Figure 1. Basic RL Structure

B. Q Learning

With the term Q-Learning we refer to a Reinforcement Learning algorithm with the following characteristics [5]:

- model-free: Evaluates the optimal policy without the need of a model of the environment and according to transition functions and rewards.
- off-policy: Evaluates the reward for each future action and updates the policy, without following a predefined behavior.

The agent uses the rewards of the environment to carry out the best possible action for the present state. In a more simplified version of the Q-Learning algorithm, this decision is taken in accordance with a Q table consisting of Q-values which match the actions with the possible state-action pairs. In the course of the training, this table is renewed, improving the Q-values that returns a greater reward. [6]

For each finite Markovian decision process, Q-Learning finds optimal policy maximising the value of the overall reward for all subsequent states, starting with the current one. It uses Temporal Differences (TD) to assess Q*(s,a), meaning the estimated discounted value, and therefore learns through its experience without knowledge of the environment.

Initially, Table Q is defined, which is the structure used to calculate the maximum estimated future reward for the action of each state. Initialisation shall be carried out at zero values with a MxN scale, with M number of s and N the number of actions α.

Then, the agent selects an action. The epsilon-greedy strategy is being implemented here. In detail, the choice of an action is initially carried out randomly. Gradually, and while agent acquires a better picture of the environment, epsilon decreases, so agent becomes greedier, choosing only actions that will certainly bring him a great reward, so it involves less exploration of the area. The balance between exploitation and exploration is very important. With greater exploration, the agent will wander aimlessly in the space following tactics that do not help him achieve the ultimate goal, and with great exploitation, the agent is in danger of being locked up at a local minimum.

After the execution of the action, the agent receives the reward from the environment and updates the Q table.

The values of Q table are updated through Q-function, which uses the Bellman equation taking as inputs the state (s) and the action α. [7]

$$\hat{Q}^{\pi}(s_t, \alpha_t) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Rt + 3 + ...|s_t, \alpha_t] \quad (1)$$

C. Deep Q Learning

Deep Q-Learning algorithm follows the same logic as Q, except it implements the use of neural networks instead of a Q-table to estimate Q-values. In such a network, input is the current state and output are the Q-values for each of the actions. [8], [9]

For the implementation of such a network, the first key step is to store in memory the previous experience of the agent. This process is called replay memory:

$$e_t = (s_t, \alpha_t, r_t, s_{t+1}) \quad (2)$$

The reason that replay memory is used is initially to make agent experience more efficient as it is reused during the course of training. This allows the system to learn from concrete examples multiple times. In addition, it helps not to ”forget” past experiences but also to reduce the correlation between similar situations.

The next step concerns the training of the network. According to the equation (2), the network’s loss of function can be calculated by finding the difference between Q value and the desired Q. In more detail, using the Bellman equation, the desired Q-value (Q-Target) is assessed:

$$Q_{target} = R_{t+1} + \gamma max_a Q(S_{t+1}, \alpha) \quad (3)$$

$$Q_{loss} = R_{t+1} + \gamma max_a Q(S_{t+1}, \alpha) - Q(S_t, A_t) \quad (4)$$

An important challenge presented in line with the above logic is that the system uses the same parameters for the assessment of Q-target but also Q-value, resulting in each step of education, both values being moved. A solution is the implementation of different neural networks, with the Q-Target estimation network to have stable parameters values, and to be renewable by Q-value network after C steps.

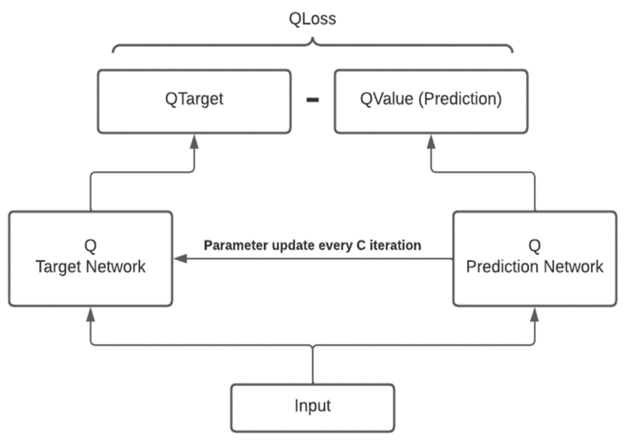

Figure 2. Different network implementations for Q-Loss

D. *NeuroEvolution of Augmenting Topologies (NEAT)*

The term Neuroevolution defines the Machine Learning method by which evolutionary algorithms are applied for the manufacture of artificial neural networks. Applications with this method over reinforcement learning problems have high levels of effectiveness, as compared to traditional methods of learning with stationary neural networks, they have a high generalisation that allows learning without clear objectives and with little feedback. A big question in Neuroevolution is the simultaneous evolution of the topology of neural networks by training the weights of connected neurons. In need of overcoming this challenge, the NEAT algorithm, which belongs to the TWEANNs group (Topology and Weight Evolving Artificial Neural Networks) was developed. NEAT is showing that it outmatches evolutionary algorithms of a specific topology, that is, a very structured form of neural network, which only evolved the values of the trainable weights. [10] The basic idea of NEAT revolves around addressing the problems faced by TWEANNs and how the algorithms was designed to deal with them. [11]–[14] These technical difficulties that emerge are:

1) Finding a genetic representation of networks that allows different topologies to apply crossover to each other.
2) The protection of topology, which takes a few generations to optimise, so that it does not disappear prematurely from the population, as the ultimate purpose is to find a simple topology that can only benefit from added complexity.
3) The minimisation of topologies during evolution without the use of a fitness function that calculates the complexity.

To solve the first challenge, the networks concerned are traced back to a representation called genome and the corresponding expression of the network in a form called phenotype. Each of the genomes includes a list of connecting nodes that describe:

- The in-node
- The out-node
- The weight of the node
- The enable bit which indicates the existence of a node in the network
- The innovation numbers

The innovation number is particularly important for the creation of new networks. For each new connection node (gene), we set an innovation number which is unique across the entire range of the training. The creation of new networks can be done through one of the following methodologies.

1) Mutation: The network mutation can be either on an existing network or adding a new structure to it, while at the same time changing the structure of the network but also the weights of the connection nodes. Structural mutations are implemented in two different ways:

- by adding a connection where there is a merging of two existing nodes
- by adding a node where a new node is created and the two corresponding connections of input and output

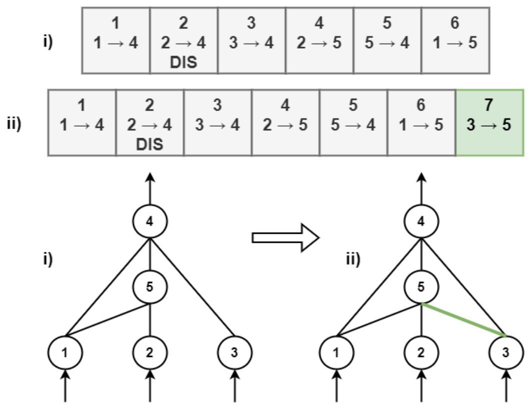

Figure 3. Genotype and Phenotype example of mutation by adding a connection

2) Crossover using innovation number: A major problem with crossover between networks is that it can create very complex or even non-functional networks due to a completely different structure and size. In the algorithm NEAT, this is easily resolved using the innovation number, a track number given after each new development. Initially, for crossover to be applied between two genomes, the algorithm needs to know which genomes in the population are from a common ancestor. These can easily be found using the innovation number of each gene, which is unique. Thus, the origin of each gene in the system is known and we can easily find the pairs to be joined together, whereas genes that do not match are dismantled or concatenated.

In this way, NEAT applies crossover between two different networks without adding unnecessary complexity to the network or charging it with computational costly topological analysis. At the same time, any network structures can be merged without further analysis and in a rather simpler way.

3) Speciation: A major handicap presented during evolution is the fact that the new structures in their creation have

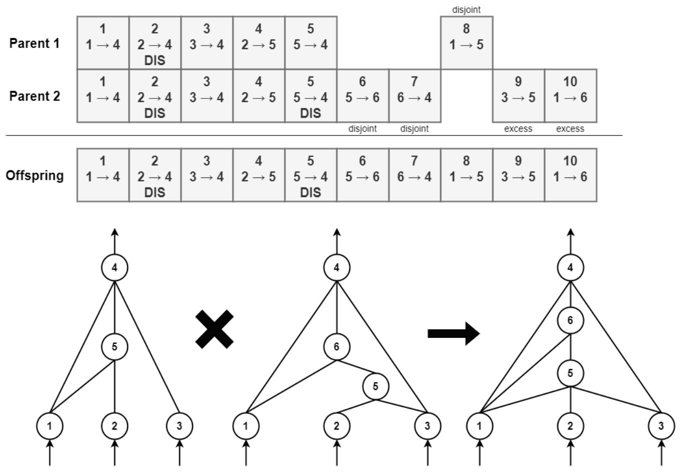

Figure 4. Genotypes and Phenotypes example of crossover

an increased error. By adding a new connection or a new node without any optimisation of the network's weights, it would lead to a significant disadvantage vis-à-vis other networks of the population. One way of dealing with this problem is to protect these new networks by categorising the population into species based on their topological similarity (speciation). By this method, new networks are predicated to optimise their weights before they need to be compared to the rest of the population.

The seperation of networks is being implemented with the help of innovation numbers. The greater the number of non-like genes between two different genomes, the more disparate they are. The smaller the compatibility distance δ of two different structures, the more similar these structures are.

$$\delta = \frac{C_1 E}{N} + \frac{C_2 D}{N} + C_3 * W \tag{5}$$

The genomes belonging to the same category are represented by a random genome from this category of the previous generation. If a genome does not belong to any category, then this genome creates a new category with the same as a representative.

4) Minimizing Dimensionality: Based on the overall concept of TWEANNs, the population is initialized with various random topologies so that the principle of diversity exists from the beginning. Instead, NEAT starts from a small-scale space without hidden nodes, and gradually increases the complexity of its structures, thus reducing the amount of training and increasing its efficiency compared to TWEANNs.

III. Implementation

In the present work an environment simulation of a chase game between two teams of agents has been carried out. In particular, the blue team has a certain number of movements depending on the dimensions of the area in which they operate, until it catches the rival red team. Similarly, the red team in the same space should avoid blue.

The area where agents act is a grid of multiple dimensions. Internally, the environment can also include obstacles either in the form of walls or covering entire cells. The experiments were developed in three different areas for variety. Those areas are 5x5, 10x10 and 10x10 with obstacles.

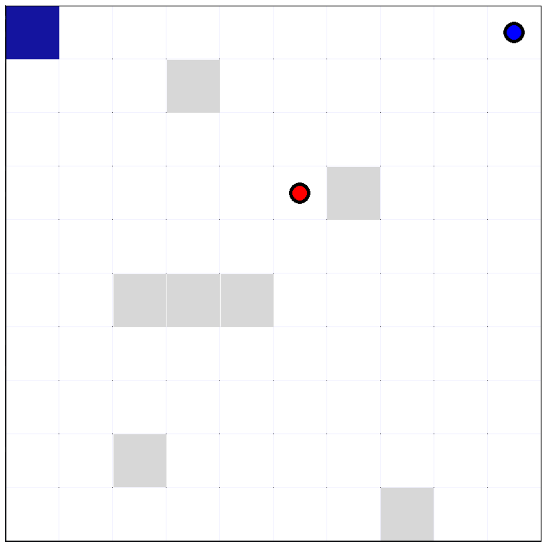

Figure 5. Sample Simulation Environment 10x10 with obstacles

The creation of the simulation environment was made using the Python 3.9 programming language. For the development of the graphic interface, the pygame library was used, whereas the OpenAI library was used for the environmental structure. The development of the code and learning algorithm was developed using keras, rl and the neat-python library. [15], [16]

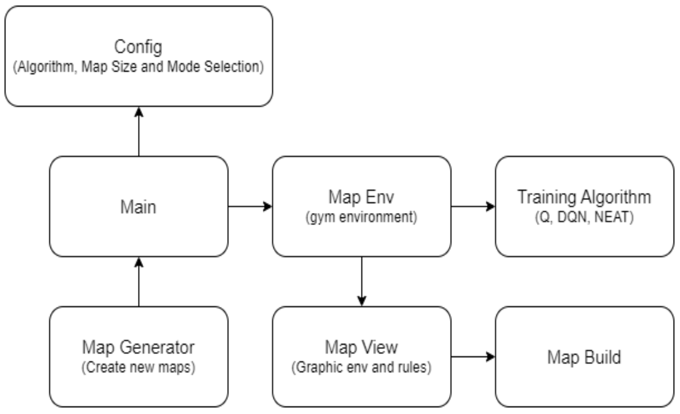

Figure 6. Simulation Code Development Diagram

For each of the above algorithms, several increased complexity experiments were carried out to draw conclusions as to their comparison at each stage of the training. The phases of the experiments are as follows:

Experiment Phases:

- Static: Training the blue team to catch the red, which changes random positions and stays still in every game.

- Random Movement: Blue team training to catch the red one. The red team randomly moves in every step of the episode.
- 2Players: Simultaneous training of both teams.

Formatting the input and output into each algorithm is subject to a pre-processing. Initially, the input can be expressed in the positions of the two groups after the necessary pre-processing. At the phase of the experiments, input was given coded, either in one-hot encoding or normalized to 1, the coordinate values for each group. After the experiments phase, no change was made in relation to the coding, so input into the system was given as it stands.

A challenge that had to be addressed is the behaviour of the agents with the obstacles. A thought about the recognition and treatment of obstacles is to add the positions of the obstacles to the input, as far as obstacles occupying a whole cell are concerned. Both groups needed more input to be able to interact with them, thus raising the space and time of education.

An important parameter for conducting experiments is to maintain the reward for each phase of the problem. The reward given to the system for each of its action and its next state is to be the same in every different methodology.

Finally, an issue that rose during the experiments was the fine-tuning of the hyper parameters of each algorithm. Their selection was decided after many tests and changes, particularly between the common hyper parameters like the number of episodes, number of consecutive successes to end the training etc.

Table I
Q RESULTS

Map	Time	Episodes	Max Reward	Problem
5x5_empty	00:00:41	226	100	Static
	00:08:23	935	95	Random
	00:45:23	10586	80/20	2Players
10x10_empty	00:28:38	3532	85	Static
	07:12:01	25454	75	Random
	-	-	-	2Players
10x10_obst	00:31:15	3951	85	Static
	08:18:45	25810	70	Random
	-	-	-	2Players

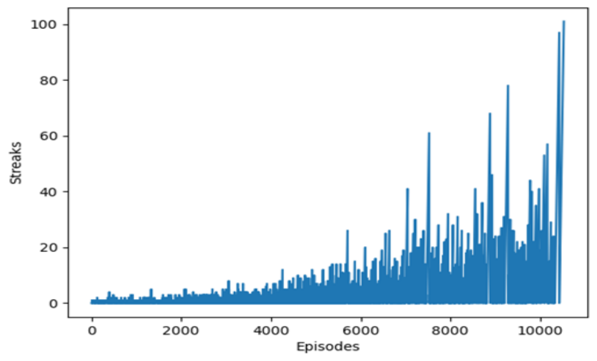

Figure 7. Consecutives successful tries (Streaks) per episode

IV. Experiments and Results

A. Q-Learning

One of the main reasons Q-Learning was a candidate for the following experiments was its finite and relatively small, depending on the experiment, area of the environment. Therefore, the table of Q-values to be created will be small, so the training time was small compared to other learning methods. If the area grew larger, the training time increased exponentially. So, in small areas like 5x5, Q had quite good results, but by increasing the complexity and the scale of the environment and consequently the Q-table, the algorithm could not generalize to an optimal solution of the problem. This becomes quite evident in Table I. The times in the problems of two simultaneous training expresses the end of the first save of the first successful training.

In the experiment carried out by both groups (2Players), the same logic developed in each step for each of the agents. Two Q-tables were created, one for each agent, which in the testing phase took turns acting. However, in more complex environments, it failed to converge. For example, in the area 10x10 without obstacles to the 2Players experiment, the maximum observation state for each agent is [10, 10, 10, 10] and action [4]. So, we end up training two Q-tables of 10000x4. In this example, the algorithm failed to reach a solution of up to 30000 episodes. This leads us to the conclusion that even if we increase the number of episodes and achieve the desired solution, it will hardly be able to generalize in experiments of increased complexity or larger areas.

The first conclusion resulting from the completion of all experiments was the ineffectiveness of the simple Q in complex problems. In addition, increased computing power systems are required to record relevant experiments as the training of such an agent can last even days.

In conclusion, Q is a quick and effective algorithm to solve simple structured problems. By increasing complexity, however, resorting to other methodologies is necessary.

B. Deep Q-Learning

During the experiments with Q-Learning, many weaknesses were observed in training, mainly due to the exponential growth of time but also the size of Q-table and the complexity of the experiments. The solution to these weaknesses was the conversion of Q into Deep Q. Replacing the Q-table with a Dense neural network greatly reduced the learning area and the number of weights to be trained. [17]

As far as the 2Players experiment was concerned, two substations of the above implementation were created and were trained in different times. First became the training of the blue team and then the red team. While red was training, the blue moved according to the model previously trained.

Learning was not done at the same time as suggested in Q methodology. The reason why they were selected to take individual training is the obvious head start of the red group during the simultaneous training, so that the blue team does not manage to generalize properly.

The main differences between the static structures of the networks were not affected by the size of the area. The complexity of the network was primarily affected by the goal of the problem, which helped to reduce the amount of networks, depending on the experiment to be carried out.

In more detail, the network structures for the blue group are set out below.

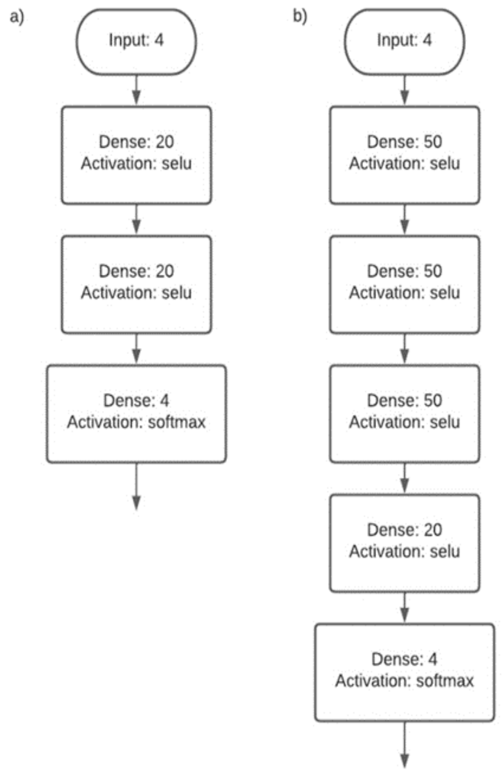

Figure 8. Neural Network Structures of Blue team a) Static, b) Random + 2Players

A major problem emerging at the end of the experiments is the difficulty of finding suitable networks but also the need to modify them until the best results are obtained, in combination with the choice of appropriate parameters.

Regarding Deep Q, experiments certainly were better overall and vindicated its choice as an improved version of the simple Q-Learning algorithm. By increasing complexity, experiments ran into a reasonable time and comparatively much smaller than Q. An important factor was the fact that no table was made during training, but a network. The number of weights is much less than the size of a Q-table, so it makes it very much more trustworthy in testing. For example, the results of the experiments, as well as their training times and maximum reward per episode, are presented.

Table II
Deep Q results

Map	Time	Episodes	Max Reward	Problem
5x5_empty	00:02:34	7	98	Static
	00:18:56	13	89	Random
	00:28:21	12	90	2P Chaser
	00:28:21	17	212	2P Runner
10x10_empty	00:07:15	10	95	Static
	00:27:55	26	86	Random
	00:45:16	14	89	2P Chaser
	00:45:16	89	157	2P Runner
10x10_obst	00:15:10	20	94	Static
	00:58:13	29	76	Random
	02:56:17	26	79	2P Chaser
	02:56:17	54	221	2P Runner

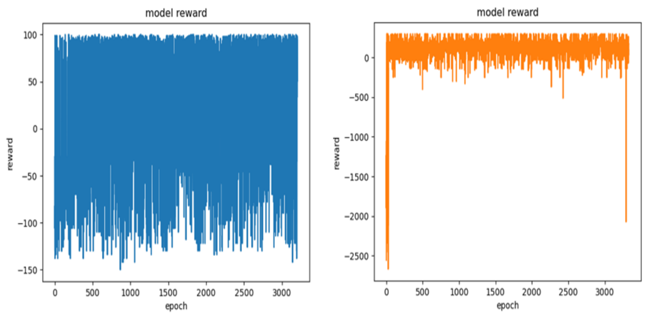

Figure 9. Model Reward Chaser-Runner during training at 10x10 area

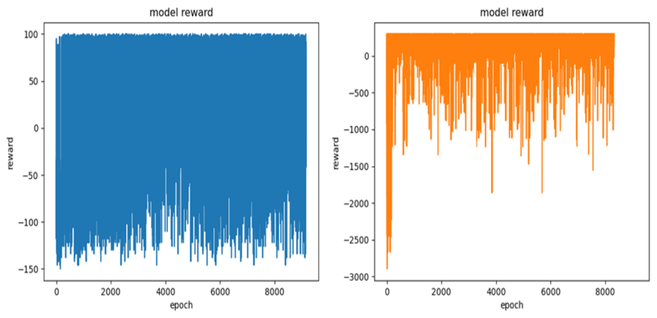

Figure 10. Model Reward Chaser-Runner during training at 10x10 area with obstacles

In Table II, Max Reward expresses the maximum reward by the average agent per episode. According to the rewards given, the most likely reward for the blue team is 100, whereas the red team is 250. In experiments there was a slight superiority of the chaser, the blue team. This may be due to the use of the same network structure for both agents.

The goal of the red group is more difficult as it is to face an already trained system, so a more complex model would have better results.

In addition, according to the above figures, the mean reward per epoch is presented, which expresses steps to achieve the target or exceeding the step limit. It is noted that the models actually generalize a lot faster than the epochs they ran into. This may also be due to the randomness of the examples. A failsafe to find the best model is a call-back implementation, a process that is embedded in the fit function of education and stores the best model after each episode until that moment.

C. NEAT

The latest methodology implemented was the NEAT algorithm. Theoretically, NEAT is the most promising learning algorithm from all three. This is because it overruns in speed and volume from Q but also allows using neuroevolution the development of optimal, or near optimal, neural network topology. The second reason is also a great advantage in relation to Deep Q, which the basic structure of its network relates to a static structure which has been found after many tests and experiments of different topologies.

As in previous experiments, in the 2Players, training has been carried out separately and simultaneously. Finally, separate training was selected for the same reasons as the implementation of Deep Q.

Table III
NEAT RESULTS

Map	Time	Generations	Fitness Result	Problem
5x5_empty	0:00:09	10	98.2	Static
	0:08:59	752	93.7	Random
	0:14:45	761	92.7	2P Chaser
	0:14:45	520	215	2P Runner
10x10_empty	0:11:02	851	94.2	Static
	1:28:56	1121	87.6	Random
	2:10:12	1242	86.2	2P Chaser
	2:10:12	810	180	2P Runner
10x10_obst	0:32:04	2584	85	Static
	3:45:13	3650	82.4	Random
	4:56:30	3820	81.2	2P Chaser
	4:56:30	2110	204	2P Runner

In comparison with previous algorithms, NEAT produced more satisfying results. Finding the appropriate network structure was certainly a lead towards Deep Q which was trained in a stable structure. Overall, however, the training time was obviously greater than the rest, but not prohibitive.

The above results were achieved following a few changes in hyperparameters and tests. It is noted that for Fitness Result of the blue group, for each problem, other than ”2Players Runner,” the maximum fitness to be achieved is 100. Similarly, for the red group and ”2Players Runner,” the maximum fitness is 250 as it gets 10 reward for every valid move to a maximum number of movements 25.

It can be noted that as long as the area gets difficult, the chaser's fitness is decreasing. On the contrary, the runner has a fairly good effect, which is obvious as the chaser has not been sufficiently trained to catch the runner in each episode. In addition, the runner appears to have a lead with the addition of the obstacles.

The effectiveness of NEAT, particularly to the chaser, has led to an effort to improve the results, seeking optimal parameterisation. A very important change was the diversification of the way in which the reward was awarded. In more detail, for the runner, tests were carried out with the award of a reward = -100 when caught by chaser and reward = 4 for any other movement. In this way, more emphasis is placed on avoiding the opponent even if the movement that the agent is doing is not valid. At the same time, the maximal fitness of the runner changes from 250 to 100, as it now achieves a reward = 4 for a maximum number of steps 25. Thus, according to Table IV, we achieved 8% improvement of fitness result for the chaser and 4% improvement for the runner.

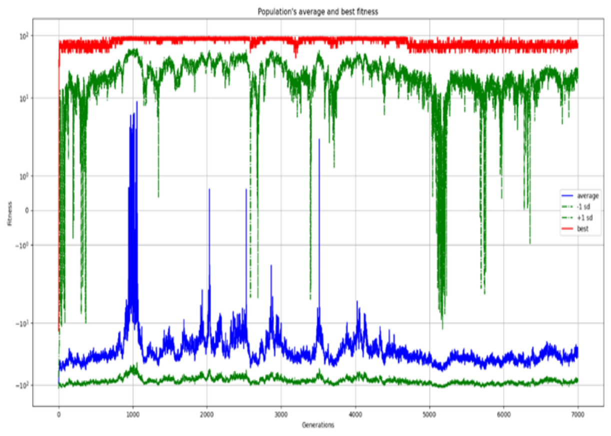

Figure 11. Mean and Max Reward during Chaser training

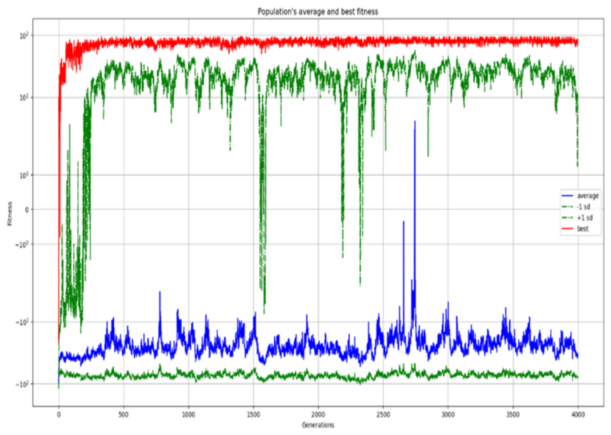

Figure 12. Mean and Max Reward during Runner training

Table IV
NEAT RESULTS AFTER OPTIMAL PARAMETERISATION

Map	Time	Generations	Fitness Result	Problem
10x10_obst	4:21:40	1081	89.5	2P Chaser
	4:21:40	2412	85	2P Runner

V. CONCLUSION

The general picture presented by the experiments shows the superiority of NEAT across the other algorithms in relation to this problem. Nevertheless, the low processing power in which experiments have been tested, in conjunction with the complexity of the experiments, makes it difficult to set up more complex networks created by neuroevolutionary sequence. Thus, by implementing experiments in systems of greater processing capacity, we would have been leading to better results and therefore in smaller training times, even in implementing experiments such as Q algorithm.

This work was a learning approach using three different algorithmic logics. The results that emerge are desirable, but the relevant literature on similar techniques is very large. Other relevant learning pathways could be used in future research that can deliver the best results.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[3] H. Kung-Hsiang, "Introduction to various reinforcement learning algorithms," *Part I (Q-Learning. Towards Data Science. Retrieved from: Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)— by Kung-Hsiang, Huang (Steeve)—Towards Data Science*, 2018.

[4] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[5] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99. Citeseer, 1999, pp. 278–287.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[7] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE access*, vol. 7, pp. 133 653–133 667, 2019.

[8] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.

[11] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[12] ——, "Efficient evolution of neural network topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2. IEEE, 2002, pp. 1757–1762.

[13] S. Lang, T. Reggelin, J. Schmidt, M. Müller, and A. Nahhas, "Neuroevolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies," *Expert Systems with Applications*, vol. 172, p. 114666, 2021.

[14] P. Bertsekas Dimitri and N. Tsitsiklis John, "Neuro-dynamic programming," *Athena Scientific*, 1996.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[16] A. McIntyre, M. Kallada, C. G. Miguel, C. Feher de Silva, and M. L. Netto, "neat-python."

[17] A. Choudhary, "A hands-on introduction to deep q-learning using openai gym in python," *Analytics Vidhya*, 2019.