Simulation and Comparison of Reinforcement Learning Algorithms

Konstantinos Spyropoulos [1]
Dionisios N. Sotiropoulos, PhD [1]

July 2, 2023

[1] University of Piraeus, Department of Informatics

Table of Contents

Introduction

Reinforcement Learning

Reinforcement Learning is the field of Machine Learning which specializes in creating agents which operate and interact in a dynamic environment. 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.

Problem Definition

Problem Definition

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.

Problem Definition

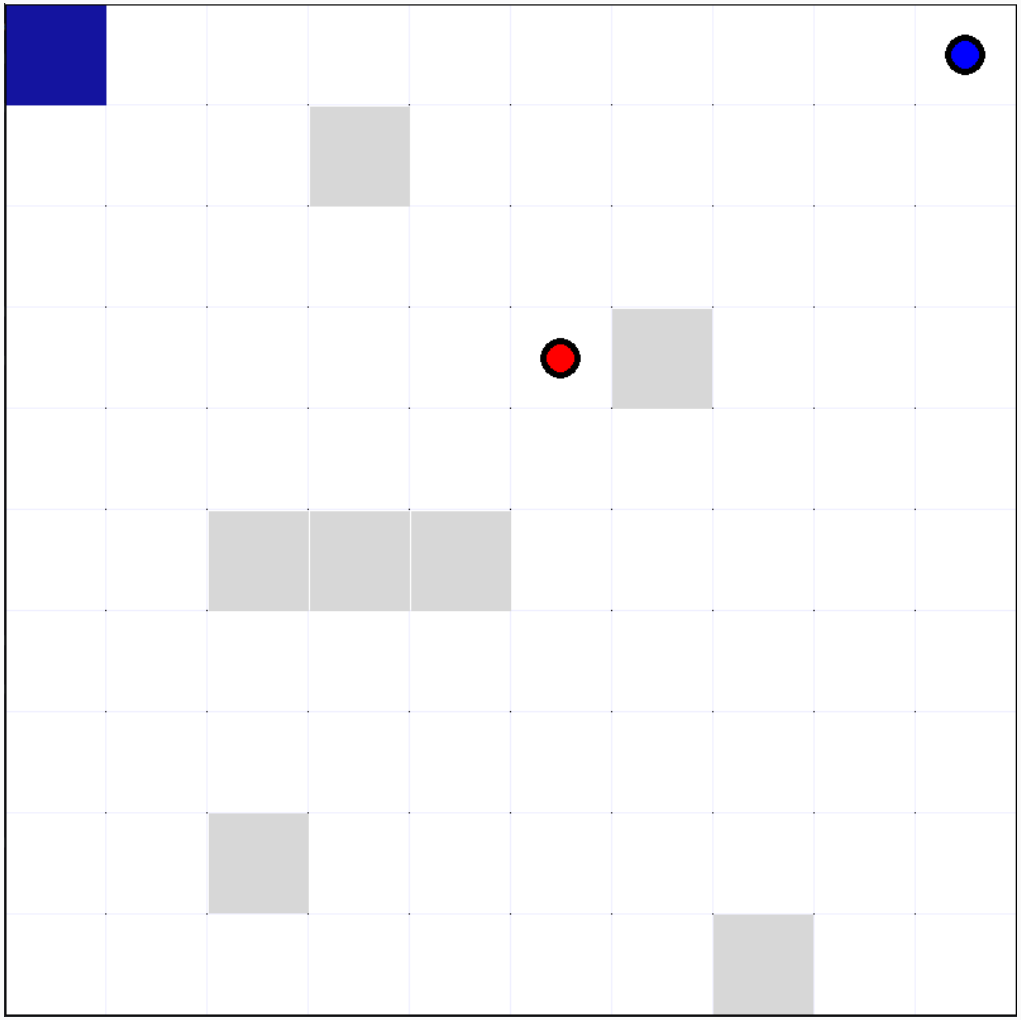

Figure 1: Sample Simulation Environment 10x10 with obstacles

Methodology

RL 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.

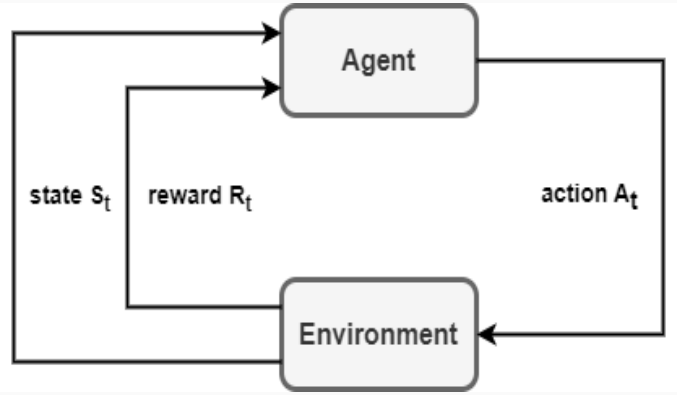

Figure 2: Basic RL Structure

Methodology 1 - Q Learning I

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

- **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.

Methodology 1 - Q Learning II

The agent uses the rewards of the environment to carry out the best possible action for the present state. This decision is taken in accordance with a Q-Table consisting of Q-values which match the actions with the possible state-action pairs.

Initialisation of Q-Table shall be carried out at zero values with a MxN scale, with M number of s and N the number of actions α. In the course of the training, this table is renewed through Q-function, which uses the Bellman equation taking as inputs the state s and the action α.

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

Methodology 2 - Deep Q Learning I

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.

Methodology 2 - Deep Q Learning II

Key Features of Deep Q Learning Algorithm

- **Replay Memory** Replay memory is used is to make experience more efficient as it is reused during the course of training by storing in memory the previous experience of the agent, helping it not to "forget" past experiences but also to reduce the correlation between similar situations.

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

- **Loss Function** Using the Bellman equation, the desired Q-value (Q-Target) is assessed:

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

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

Methodology 3 - NEAT I

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.

Methodology 3 - NEAT II

A big question in Neuroevolution is the simultaneous evolution of the topology of neural networks by training the weights of connected neurons. NEAT is overcoming this problem and 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.

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.

Methodology 3 - NEAT III

New networks can be created through one of the following methodologies:

- **Mutation**: Adding new connections or nodes.
- **Crossover**: Combination of two parents to create an offspring (using innovation number).
- **Speciation**: Categorizing population into species based on their topological similarity to avoid creating new structures with increased error.
- **Minimizing Dimensionality**: NEAT starts from a small-scale space without hidden nodes, and gradually increases the complexity of its structures, thus reducing the amount of training.

Implementation

Environment and Experiments Specifications

The experiments were developed in three different areas for variety:

- **5x5**
- **10x10**
- **10x10 with obstacles**

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.

Code Development Diagram

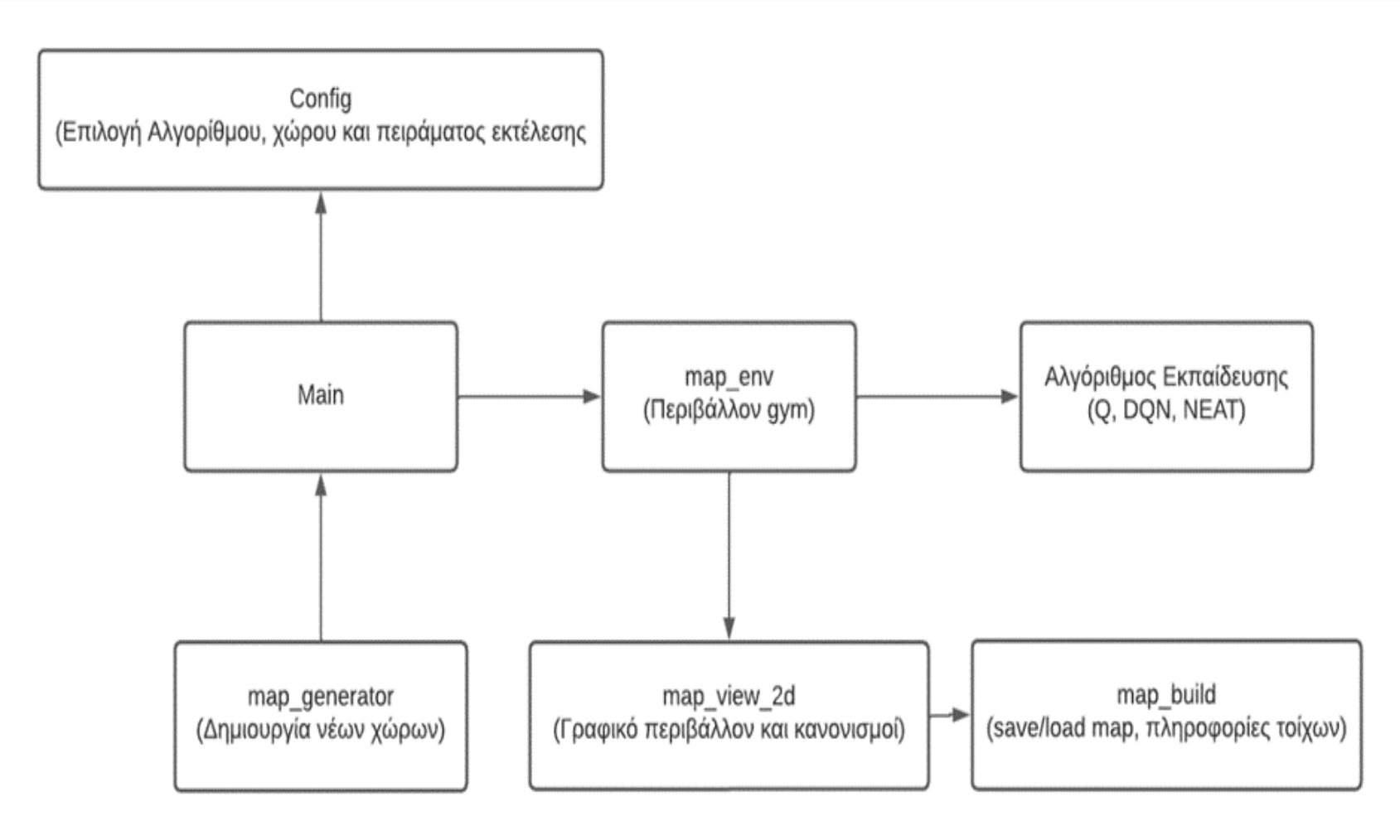

Figure 3: Simulation Code Development Diagram

Results

Q Results I

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 (2Players Experiment) and the scale of the environment and consequently the Q-table, the algorithm could not generalize to an optimal solution of the problema and it failed to converge.

Q Results II

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 4: Q Results

Deep Q Results I

Replacing the Q-table with a Dense neural network greatly reduced the learning area and the number of weights to be trained.

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.

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.

Deep Q Results II

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 5: Deep Q Results

Deep Q Results III

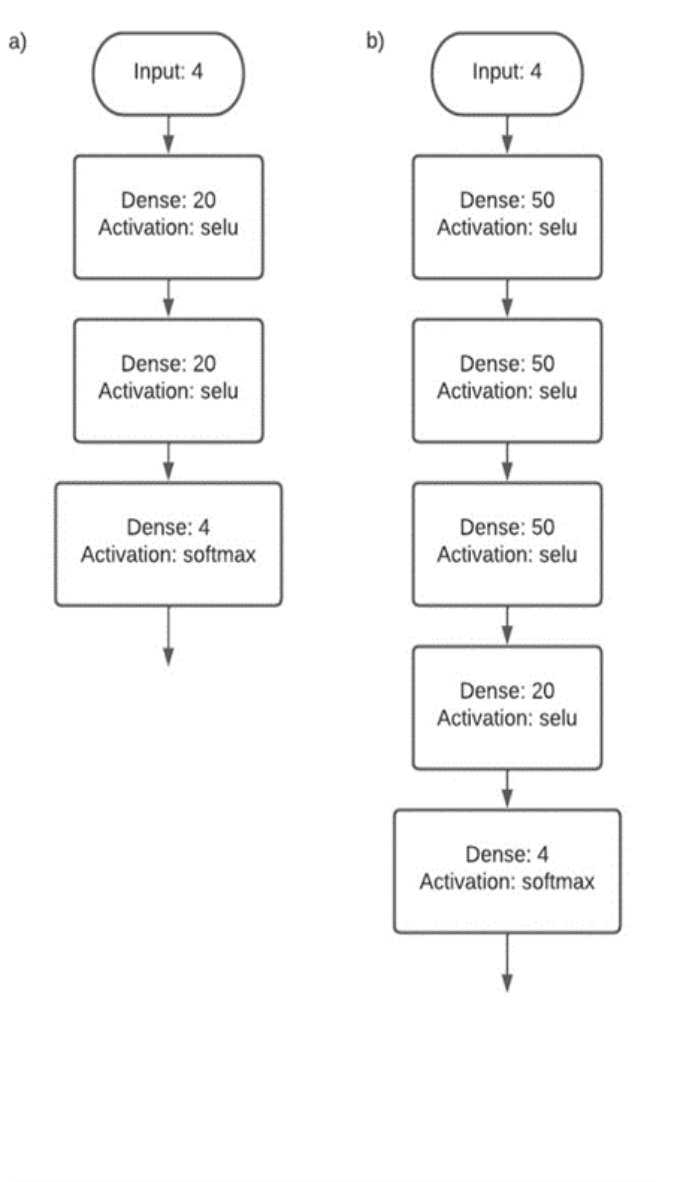

Figure 6: Deep Q Trained Models (depending on the complexity of experiment)

NEAT Results I

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 neuro-evolution 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.

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.

NEAT Results II

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

Figure 7: NEAT Results

NEAT Results III

In the below figures, the produced neural networks from NEAT algorithm are presented. More specifically, in 2Players experiment, two separated neural networks are produced, one for each agent (Runner and Chaser).

The networks are provided with the observation of the agents input, thus its current state:

- **x_agent:** X position of the agent
- **y_agent:** Y position of the agent
- **x_enemy:** X position of the enemy
- **x_enemy:** Y position of the enemy

Additionally, the networks' outputs are directions of the movement of the agent **(W, S, N, E)**.

NEAT Results III

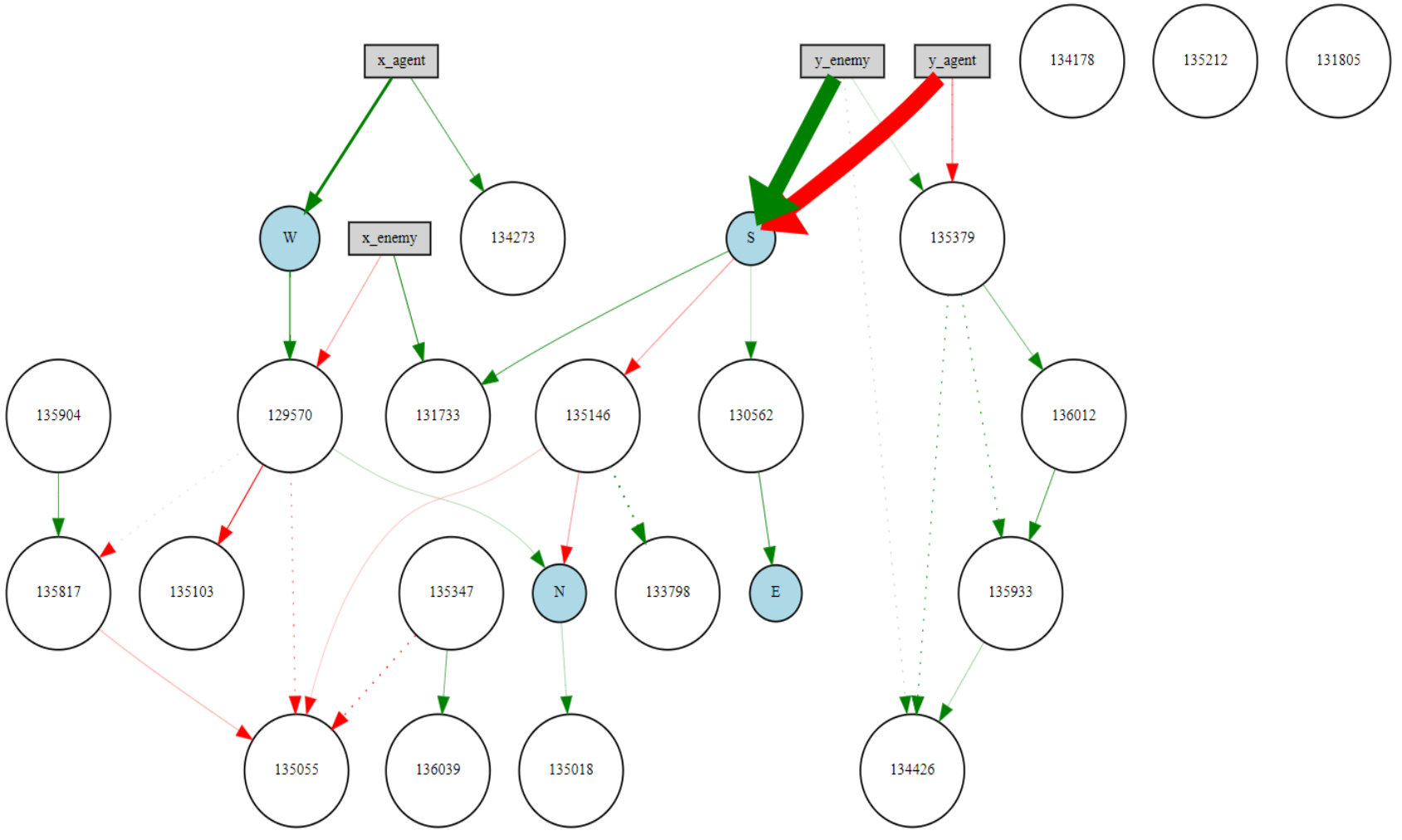

Figure 8: NEAT Runner Model

NEAT Results IV

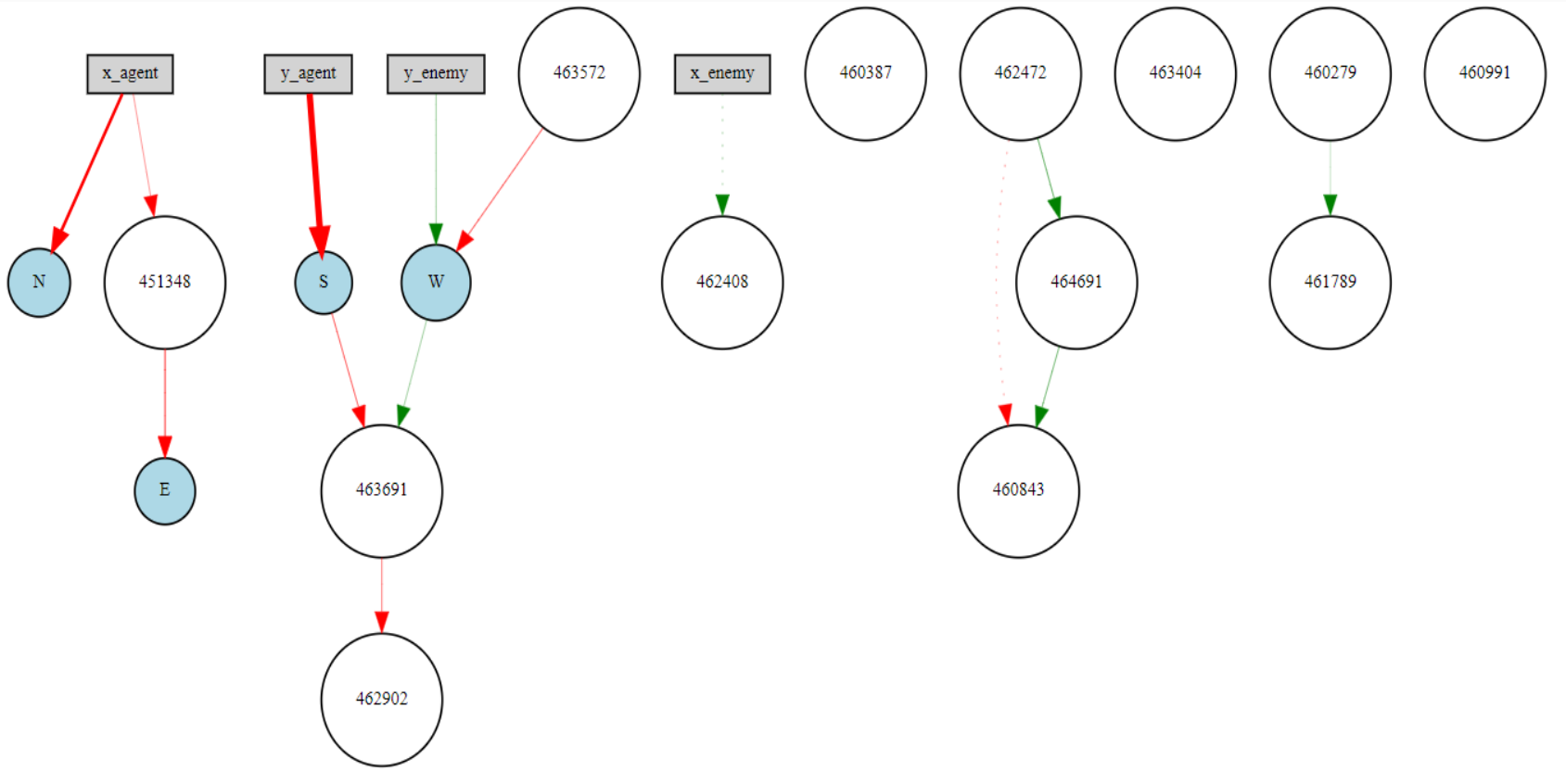

Figure 9: NEAT Chaser Model

Conclusions & Future Research

Improvements & Future Research

- Training with CNN instead of Dense Neural Networks as it shows better results in similar problems for Deep Q Learning methodology.
- Experiments with input encoding.
- Fine Tuning NEAT parameters.
- Alternative methodologies to explore (Monte Carlo, Genetic Algorithms)

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 neuro-evolutionary 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 approaches. 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.