CENG 3512 Evolutionary Computation Final Project Report

Author: Fatih Kıyıkçı, 170709032
Subject: Evolving an AI Agent to Play the Game of "Scramble Arcade"

1 Problem Statement

This project is about implementing an artificial intelligence (AI) for a game implementation of my own, called "scramble". There are many techniques on training AI agents on video games, such as Reinforcement Learning or training a neural network architecture using genetic algorithms. I used a technique called " NeuroEvolution of Augmented Topologies " (NEAT). Using this method, i trained a player that can clear out all the enemies and can get away from shots fired to the player.

2 Methodology

2.1 NeuroEvolution of Augmented Topologies (NEAT)

The article "Evolving Neural Networks through Augmenting Topologies" is released at 2002 by Kenneth O. Stanley and Risto Miikkulainen. Since the release of the paper, there have been many versions of the NEAT techniques implementation. Although there are some differences between implementations, the general idea do not change.

Working principles The general idea of NEAT, is to evolve neural networks with varying topological structures and complexities, using genetic algorithms. In the context of the NEAT algorithm, a neural network is a phenotype of the genetic algorithm. Whereas the parameters of the neural network, such as the weights, connections, or the topology is the genotype [2]

2.1.1 Encoding

The authors of the NEAT paper compares several encoding methods such as graph encoding, binary encoding, non mating and indirect encoding (Cellular Encoding in particular), the authors eliminate all encoding methods other than direct encoding, therefore the NEAT algorithm uses direct encoding. The encoding structure contains a list of neuron genes and link genes. While the neuron genes are responsible for specifying the type of the neuron, e.g input, hidden, output, the link neurons represent the connection, which specifies the connection start and endpoints, type of connection and an innovation number, which indicates the order that the connections are added.

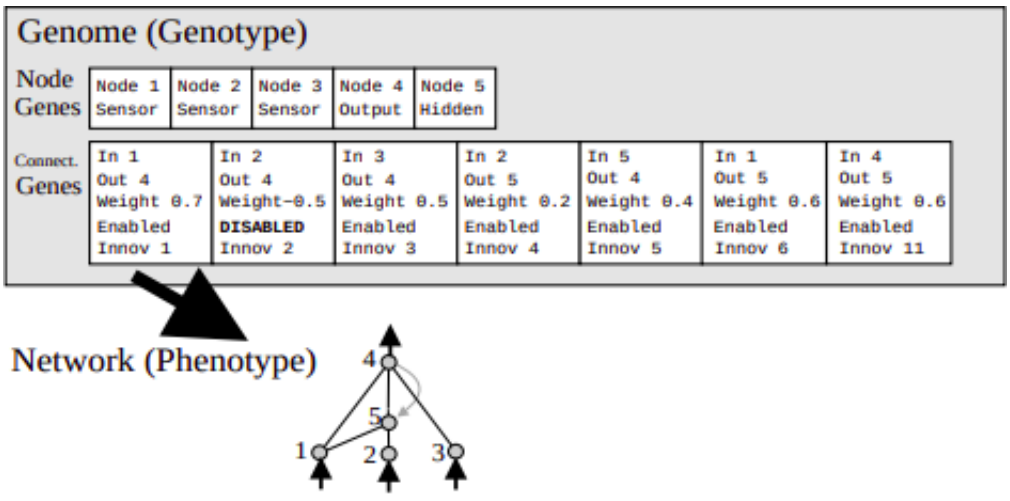

Figure 1: Encoding of connection and node genes [3]

2.1.2 Mutation

There is two different types of mutation in the original NEAT implementation. A mutation can change both the weights of connections and the structure of the network. The weights of a connection can mutate in a classical way, in which the weights can be added/subtracted a random floating point number with a certain

probability. The only difference in weight mutation is that, a weight can be completely replaced, depending on the probability. For the structural mutation, there are two types of operations. First, the mutation can add a new node to the network, then connecting an existing node with the new node. Second, it can disable a connection, add a new node in between the previously connected nodes, and enable the connection which connects all three nodes. One of the main objective of the authors was to making the evolution process go from simple to complex, this type of structural mutation where the network is mostly expands, is accelerating this objective.

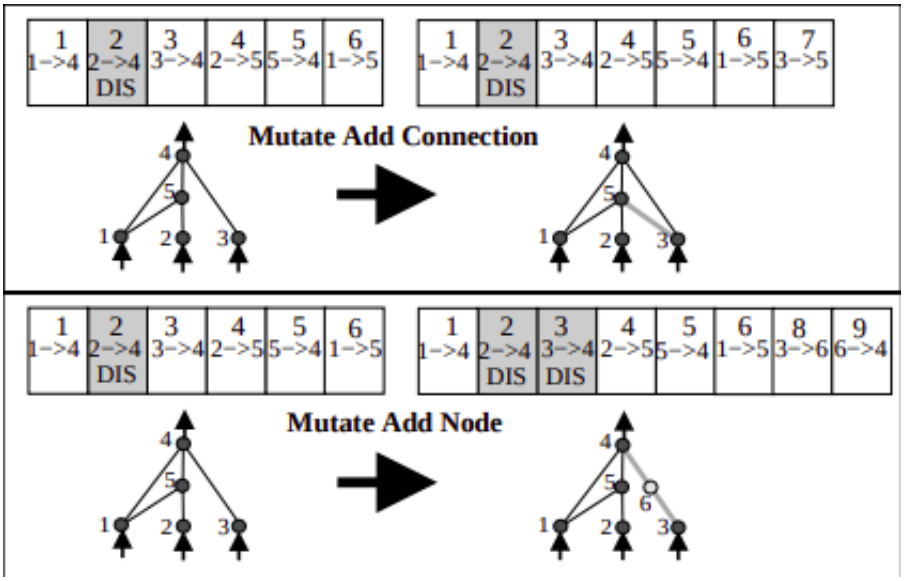

Figure 2: An examplary mutation representation [3]

2.1.3 Competing Conventions

The objective of gradually expanding the networks can be achieved by the mutation part, however in a population, different individuals can, and very likely will be vary in size. This variance of size and structure will be an obstacle in the way of crossover, since they cannot be aligned. NEAT tracks the genes historically, using the innovation number mentioned in the encoding section. This tag is tracked and whenever a gene mutates structurally, it is incremented. Therefore the history of every genome is already known in the system.

2.1.4 Speciation

Even though adding new structures to the population and with the help of historical tracking, crossing them over, creates a diverse population, because of the faster optimization time of smaller networks, large networks are rarely protected. This is solved using a method called speciation, which protects the newly innovated networks by making them compete within their own league. Speciation makes this happen by using the historical tracking.

2.1.5 Fitness Function

For experimental purposes, i have reduced the complexity of the game using many permutations, therefore some fitness metrics do not exist in some experiments. The main aim of this fitness function is to find and individual

Table 1: The parameters of fitness function

Player event	Fitness Score
Kill enemy	10
bullet fired	-1
rocket thrown	-2
died	-50
level ended.	100

that kills as much enemy as it can, do not shoot a bullet or throw a rocket for nothing, and end the level successfully.

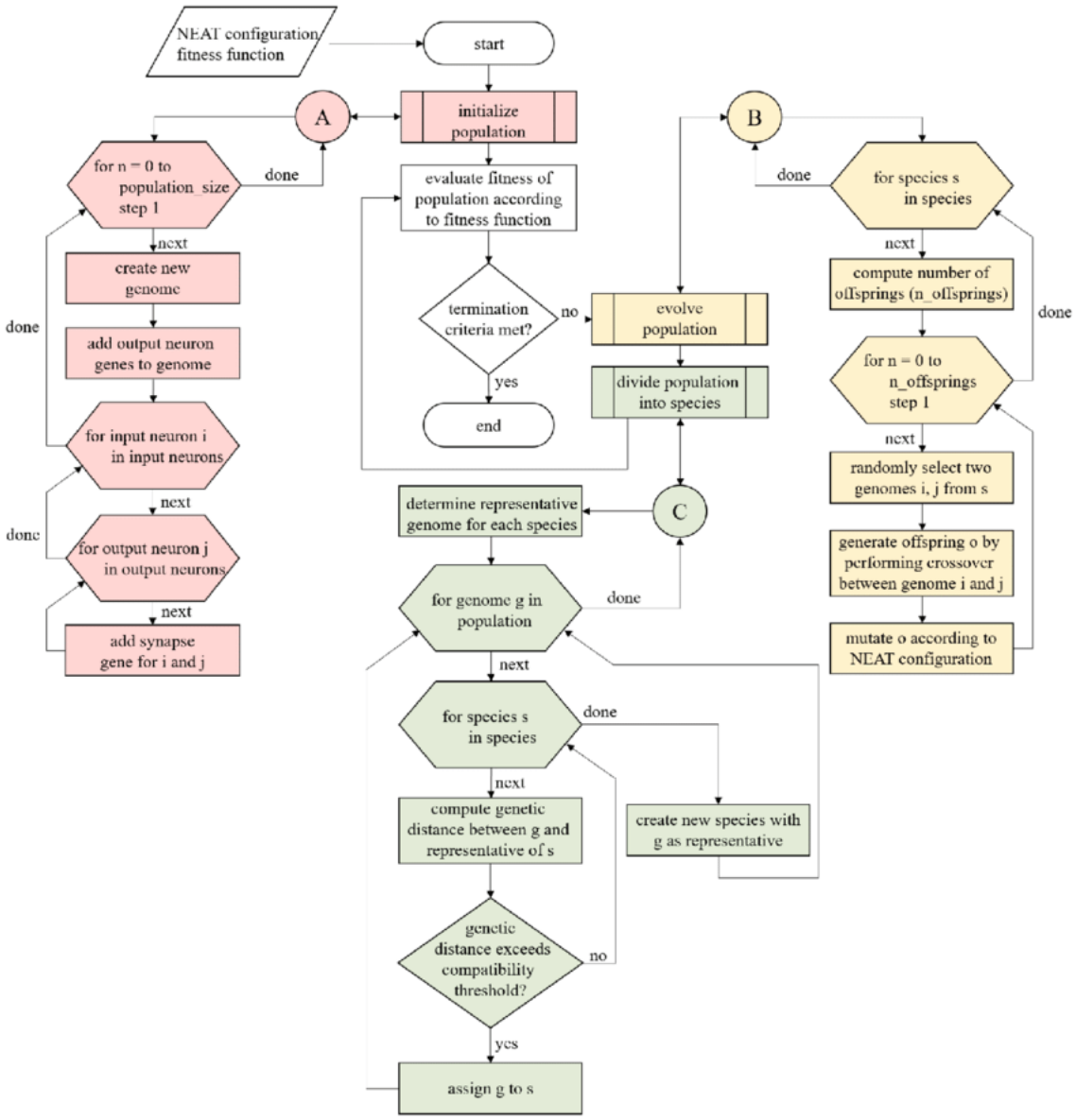

Figure 3: Flowchart representation of the neat algorithm [1]

3 Scramble Arcade Game

Scramble is an old arcade game, i have implemented this game some time ago, there are enemies heading towards the player, there is a floor with varying bumps and pits, on top of the floor, there are rockets, which go up in a particular random location. The player can shoot bullets or throw rockets, The player has a fuel point, which is reduced when a bullet is fired or a rocket is thrown. To increment the fuel, the player must collect (throw rocket at) the fuel tanks on top of the floor. The game ends when the map is finished.

3.0.1 Experiments

I have conducted many experiments, in order to find the best possible environment for the NEAT algorithm. Using the default environment of the game, and inputting the characters positions to the NEAT algorithm, did not yield a good results, the algorithm was not able to learn the objectives mentioned in the fitness function, after 30 generations, therefore i have decided to conduct my experiments, starting from the simplest and gradually adding complexities.

3.0.2 Inputs

One of the most challenging tasks of this project was to create reliable input data. I have conducted many experiments in order to find a good set of inputs. In the process, i have wrote many functions and permuted the calculations of these functions as input features. The most common functions i experimented with were:

- distances between the player and all environment objects
- rightx, leftx, topy, bottomy positions of all environment objects and the player.
- rightx, leftx, topy, bottomy positions of the closest n individuals from each object type
- intersection points of the players bullet aim and the enemies
- intersection points of the players missile aim and the enemies
- trajectory positions for the missile of the player
- top and bottom straight lines' intersection points

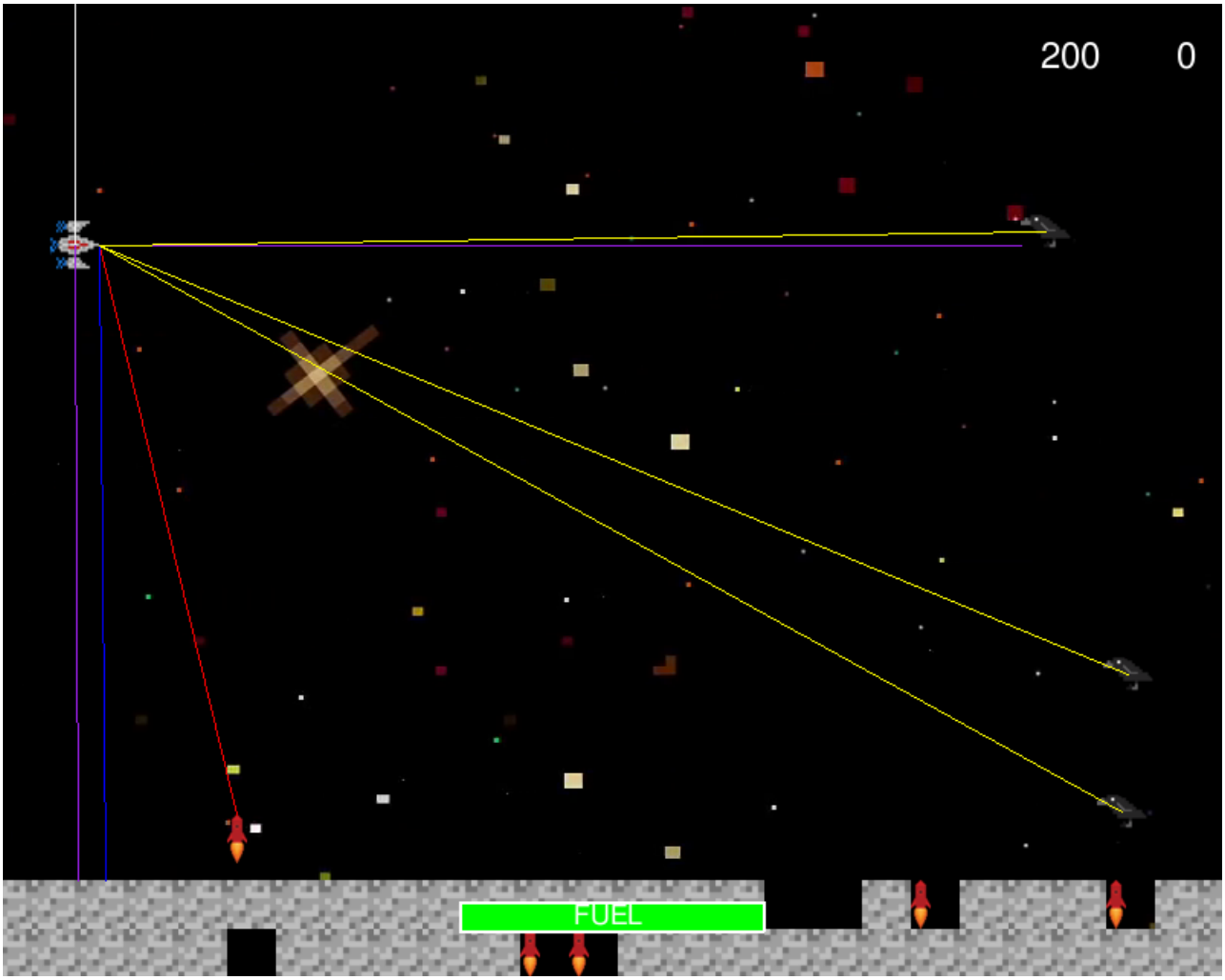

Figure 4: Representation of the inputs. Items 3,4,7

Apart from these input experiments, i have drawn a compressed version of the game environment in every frame, (20*60) and fed the neural network with the flatten version of the matrix. As i saw from other peoples' experiments, this input strategy works better than other inputs [4], [5]. However, since the input dimensions are too large and the process is computationally expensive, the evolution process is very slow. Therefore i could not get to see the full potential of this method.

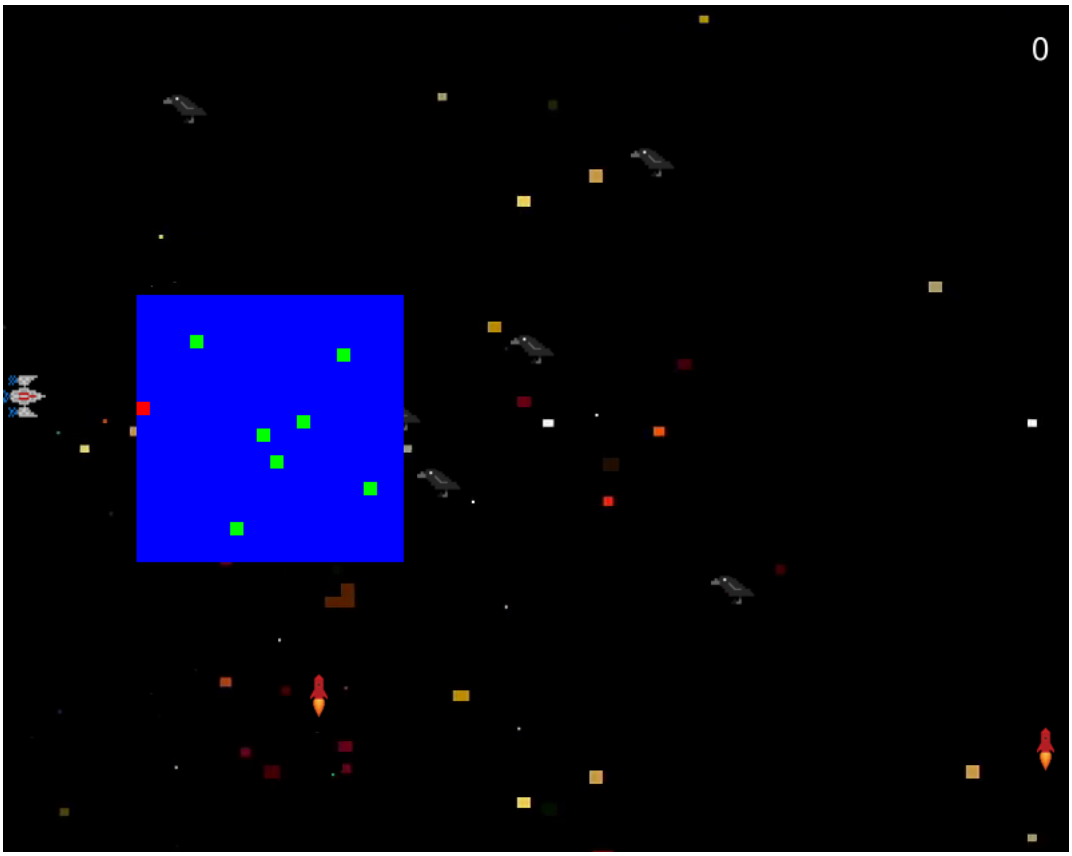

Figure 5: Compressed Image Input Representation

4 Results

The experiments i made, without simplified environments using the listed inputs (distance, positions, etc.) were not able completely play the game as i wanted it to be. And the compressed image matrix is too expensive to train within a reasonable time. However, in the simplified versions of the game (no floor below, and fuel is limitless. In exchange, the ability to fire rockets of the player is taken away) the network was able to completely destroy almost all enemies and finish the game, with a high score of 530.

4.1 Table for Experiments in the Environment

I have changed the environment using stones, the ability to throw rocket, missile enemies, fuel usage,

Table 2: The effect of environment complexity for the success of neat algorithm steady state genetic algorithm.

Parameter setting						Observations		
#	Stones	throw rocket	missiles	fuel	image Input	generation	population	max fitness
a.	×	×	×	×	×	40	50	530
b.	×	×	✓	×	✓	20	10	130
c.	×	×	✓	×	×	20	30	372
d.	×	×	✓	✓	×	20	30	396
e.	✓	×	✓	×	×	20	100	160

- As expected, the first and simplest environment is solved easily, in my experiments, 40 generations and 50 individuals per generation is an overkill for this environment.
- The second environment is almost the same as the first one. But since the image inputs dimensions are too large, and computing the image is very expensive, i was only able to run it 20 generations for 10 individuals per generation. Looking at similar experiments of other people, this method was the best for them.

- The third environment is just a little harder than the first one. Only the missiles are added, the individual with the max fitness is playing really good, but sometimes it is just stuck between the missile and enemies. In my replays, the same agent did finish the level a couple of times.

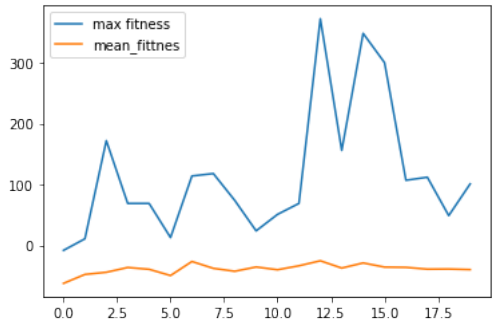

Figure 6: Third Environment Convergence graph

- In the fourth environment, The only difference from the third environment is the addition of the fuel concept. The player can shoot fuel tanks and gain fuel, and every bullet takes away 1 fuel (200 total). Interestingly, adding the fuel concept increased the maximum fitness value.

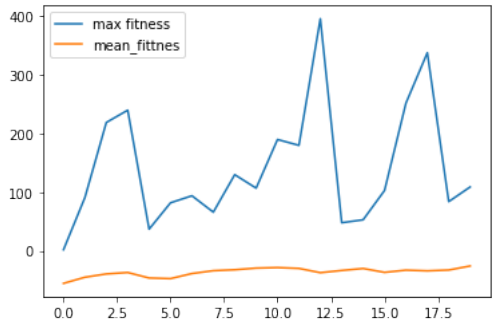

Figure 7: Fourth Environment Convergence graph

- With the fifth environment, there are floor stones added, the individuals could not evolve a strategy to avoid the floor, it may be caused by the input strategy for stone awareness, which i tried using the location of the closest stone, then the highest stone that we have ever seen in the game.

5 Conclusion

The reason i took this course was because of my curiosity towards the evolutionary algorithms, i had only seen evolutionary algorithms used on simulations and not real worlds problems. This project and my presentation paper showed that it was wrong (since my presentation topic was about the same as my thesis topic, but i use only machine learning). The course provided a way of problem solving, that i did not realized the power of. The only thing i could say may be missing from the course, was the examples of the concepts, such as a part of code, an animation, or some program. Other than this i have very much enjoyed your class, thank you for everything.

References

[1] Sebastian Lang, Tobias Reggelin, Johann Schmidt, Marcel Müller, and Abdulrahman 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*, 172:114666, 06 2021.

[2] Jessica Lowell, Sergey Grabkovsky, and Kir Birger. Comparison of neat and hyperneat performance on a strategic decision-making problem. In *2011 Fifth International Conference on Genetic and Evolutionary Computing*, pages 102–105, 2011.

[3] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[4] Vishwesh4. Vishwesh4/neat-spaceinvaders.

[5] WilliamAmbrozic. Williamambrozic/spaceshooter.