Tanks Evolvement

Genetic & Neural Network Project

Itay Lavi-Okabi

# Introduction

## Description

This project is done as a hobby – coding and playing with some AI algorithms. This project, or machine-played-game if you will, is about evolvement of “Tanks” based on natural selection.

The game is pretty simple – a certain number of tanks is set to an environment (for now – a clear window, nothing special) and start interacting with it at the same time. Interaction basically means moving and shooting.

The interactions (the game playing) last for a certain period of time, and after each game – a natural selection is made. This natural selection, though random, should give an advantage to some tanks - the higher the tank’s grade, the higher the chances it will have more “children” in the next generation (aka next game). Tank’s grade is determined by some properties that should encourage an active behaviors like survival, destroying other tanks or achieving other goals in the game.

Each tank has properties; strength of bullets, defense level, range of shooting, range of vision, etc. Each property is in some range of numbers, thus having a maximal value possible that should give both advantage and disadvantage to each wielder. Another important property of the tank is its neural network which allows it to make decision based on its state. If time and patience allow it, I’ll also include a “friendliness” property of a tank that will allow some tanks to cooperate.

As a part of genetic algorithms, random changes can occur in a tank’s “DNA” (aka its properties), making its level higher or lower. An important notice is that – to allow improvement of the neural networks, the mutating rate should be relatively low compared to the learning rate of the tanks.

To differentiate between the tanks with different properties, each property should come with some sensible visual marking – like size for defense level, turret for bullet strength, etc.

### Terminology

**Turn** – a time period when the tank can take an action.

**Interaction** – the ability of a tank to execute an action in its turn.

**Round** – a series of interactions, where every tank had at least one turn.

**Game** – the time period between the time the interactions of the tanks start until they end.

**Generation** – a game. Refers to the chronical order of the games. Can also refer to the group of the tanks in a certain game.

**N** – number of tanks in generation. Unless stated otherwise – this parameter is constant in all generations.

**NN** – Neural network.

**Zone** – the environment of the game.

**DM** – decision-making.

**Agent** – the tank in question. Meaning – a clearer name for the tank that we are talking about.

## Project Design

## Game Rules

### Rules

Outside of zone – big punishment.

### Game restrictions

To allow a simpler developing, the following restrictions were set:

1. Positioning (x & y) will be integers. Not float or double.
2. Same for angle of shooting. Assuming that, given the size of the zone, a precision of 1 degree will be sufficient.
3. Zone size is ?

### Learning process

Snake reinforcement learning > pyTorch > gradient descend

## Decision-making algorithm – pytorch

### In short

The decision-making process (DM process, or DMP) is done using a neural network. Other, easier ways for DMP may exist, but – you know – **Neural network** for G-d’s sakes. The process also should involve reinforcement learning.

The input layer is the tanks state and properties, and the output is a vector of actions to take.

For easy reading - please recall the terminology section above.

### Input layer

the DM algorithm takes several variables into account, mainly the tank’s properties, but even more importantly - the tank’s state. Meaning – the way the tank’s knowledge about the environment.

The state should be both informative – to allow good DM, but also not too detailed or memory-consuming because that will eventually make the decision-making process slower and too complicated.

Because there are many angles to which the tank can move to or shoot at, it’s not easy to come up with such state design.

#### Structure design

The input is . It should be a vector feeding the neural network, so its size should be permanent.

Agent Properties

Vector will represent the properties of the agent – except for its positioning, which is included in the next vector discussed.

The vector will be of size 3: the agents current defense level, strength level and shooting range.

Enemies Positioning

The first problem is how to represent the positions of all the other tanks. That I’ll solve by making a vector containing the positions of **all** the tanks. If a tank is out of the range of the agent, the value will be “infinity”. This assumes that the NN will be able to avoid shooting at an enemy outside of the range.

The tank should also have a general understanding of each enemy around it. Because the strength (call it “s-parameter”) and defense (“d-parameter”) of a tank is the most visible in this game, those are the only extra properties the agent will receive about its enemies.

I will make another extra “node” for future uses (“e-parameter”).

So, the vector representing the area around the agent will be (“A” for “around”) of size ; (x, y, s, d, e) values for each tank.

Zone Limits

A Boolean vector of size 4 telling whether a zone boundary is up, right, down, or left.

#### Final structure

### DM neural network

## Genetic algorithm

## Developing the game – pygame

# Easy start – Training zone

## Description

The goal is to test both the environment development and, later, the effectiveness of the DMP. We will start by creating a training zone environment for an agent – a range zone with “targets” it should shoot at, and “items” to collect. The more targets destroyed, and the more items collected – the higher the score.

## Environment developing