# Berzerk! Agents - Project 1

David Dunlap dad9489@rit.edu
Ian Ranman ixr5487@rit.edu

# Contents

# Introduction

This is an implementation and analysis of three different agents that aim to achieve the highest possible score in the 1980 Atari game *Berzerk*. *Berzerk* is a top-down game where the player must navigate through a series of maze-like rooms while battling evil robots. The player cannot touch a wall, a robot, or the bullets that are shot at the player by the robots. The player can either avoid the robots and their bullets, or they can fire back.

The agents operate in an environment where they have sensors that give them input about their environment, and they must decide on an action based on that environment. In this case, the observations are given to the agents each frame, and come in the form of a 210x160 array of pixels representing the screen state, with each pixel containing a 3x1 array of RGB values. From this input, each agent must be able to analyze this in some way to then come up with an action. This action can be one of 18 different options, consisting of no operation, firing in the direction they are facing, moving in each of the 8 directions around itself, or moving in a direction and firing at the same time. The rest of this paper will explore the three methods used, namely the A* algorithm, Neuroevolution of Augmenting Topologies, and Deep Q Learning.

# Methods

## 1.0 Neuroevolution of Augmenting Topologies (NEAT)

### 1.1 Structure

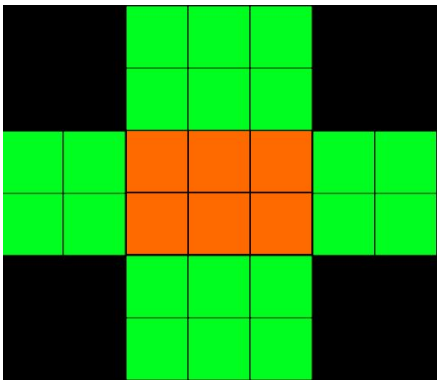
One of the agents was created using the method of Neuroevolution of Augmenting Topologies (NEAT). This is a genetic algorithm that works to evolve neural networks. The agent itself is a neural network that takes in inputs from the screen using a vision algorithm, and outputs one of the 18 possible moves. This vision algorithm is discussed further in section 1.2. The data for the neural network is stored in neat_agent/network_data.dat as a pickle dump file, and contains data about the network's inputs, outputs, weights, and biases. The network is comprised of nodes and connections that span from the inputs to the outputs, sometimes through a hidden layer of nodes. To play the game of *Berzerk*, the agent has a neural network as its "brain." It uses the vision algorithm to get a series of 80 inputs and sets the activation of its input nodes to be equal to each of these values. It then gets the activation of each node in the system using the following formula:

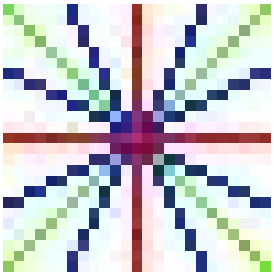
$$\sigma(bias + (response * \sum inputs))$$

The bias and response are values tweaked by the genetic algorithm, and the inputs are based on the activation of the previous node times the weight of the connection to this node. $\sigma$ represents the sigmoid function which is as follows: $\frac{1}{1+e^{-x}}$ . This is used to modulate the activation of the node to fall between 0 and 1. The move made by the network is determined by which of its 18 output nodes has the highest activation.

## 1.2 Vision

The agent uses a vision algorithm to translate the 210x160x3 array of pixel RGB values into 80 values that can be fed into the network. This is done by first downscaling the image by a factor of 2. This makes the new dimensions of the image 105x80. Next, the player center is found. This is done by looping through the array of pixels to find a pixel that is the color of the player. It then checks if this is a projectile or not. This is done by looking up, down, left, and right of this pixel. If it is clear for two pixels in that direction, it is likely to be a projectile. If it finds a pixel of the same color, it will recursively do the same for that pixel. If it ends up checking an area of pixels larger than 5x3, it is not a projectile and instead an agent as no projectile is that large. An example of the area checked when checking for a projectile is shown to the right. Once it finds a pixel the color of the player that is not a projectile, it sets the player center to be 5 pixels lower to represent the center of the player sprite. It then begins looking around. This is done by looking up, down, left, right, in the 4 diagonals, and in the 8 diagonals in-between as shown in the figure to the left. It records when it finds an enemy, a wall, an exit, an enemy bullet, or one of its own bullets. It then runs the distance between itself and the object that it found through the formula: $\frac{-1}{\sqrt{max-1}*\sqrt{distance-1}+1}$ . This is used to modulate the distance between 0 and 1, with 0 being very far away and 1 being very close. The vision fed into the network is an array of 80 values with each of the 16 directions each having 5 values representing the distance to an enemy, a wall, an exit, an enemy bullet, and a player's bullet. If an object is not found, the value is -1.





## 1.3 Learning

The network that is the brain of the agent is created using the NEAT algorithm. This is a way to genetically evolve not only the weights and biases in a neural network, but also the structure of the network itself. The networks start off very simple, with only a few connections

between the inputs and outputs. As the networks evolve, more connections are added, as well as hidden nodes in between. Each generation has 150 genomes created, with each genome encoding a network. Genomes hold unique values for each connection and node that can be used to compare genomes against each other. At the end of the generation, the networks are reproduced. This is done through crossover, where similar connections and nodes (ones with the same unique values) are crossed over. The genomes are evaluated to find their fitness by playing a game of *Berzerk*, with their score being their fitness. If the agent stands still for too long, the simulation is cut off to prevent infinite loops where the agent just stands still. The breeding and crossing over of the genomes with the best fitness, combined with random mutation, is used to create a new set of 150 genomes which are then evaluated and bred. In addition, NEAT has a representation of species to support genetic diversity. If two genomes are deemed to be different enough, they are classified as being members of separate species. Members of a species are only compared against each other which supports the evolution of wildly different structures simultaneously within the same generation. Through these techniques, the brain of the agent was trained over the course of a few days to create a complex neural network that can play *Berzerk* reasonably well.

# 2.0 Reflex Agent

## 2.1 Basic Behavior

When developing the reflex agent, several key aspects if the game were taken into consideration. First and foremost, the agent must attempt to fire and kill enemies, not only to score points, but also to protect itself. This process relies on the vision mentioned in section 1.2. The agent begins by looking for bullets within its line of sight. For this, the player center is shifted up and to the right to the approximate location of the agents arm. If any enemy bullets are found within the line of sight, the agent fires at the closest one. If no bullets are found, the agent then looks for enemies, and proceeds to fire at the closest one if any are found.

However, it does not make sense to fire if no enemy bullets or enemies are found, or if the agent currently has a bullet in the air (the agent cannot fire again until its bullet is destroyed). This leads to the agent's next most important goal, which is to reach the exit. The program will proceed to run the A* algorithm to determine the shortest path between the agent and the closest exit center. The agent will then move in this direction.

## 2.1 Optimizations

There exists a number of small optimizations to help out the agent. The first is that when choosing a moving action (i.e. an action that results in a translation of the agent's position), the agent alternates between an action that will move it closer to the goal, and an action that will move the agent in a random direction. This helps in two ways. One, the agent's line of sight for detecting enemies and enemy bullets is moved, allowing the agent to see threats that it may have

not seen in the previous frame. Two, the acting of moving toward an exit is prolonged, which can sometimes prevent the agent from moving into an enemy that is not in its line of sight (this is especially helpful when the agent is waiting to shoot again).

The next optimization is for the directions for looking for a threat. The agent first checks up, down, left and right, because enemies can shoot in these directions. If no threats are found in these directions, the agent continues to look in the other lines of sight.

Perhaps the most important optimization has to do with the A* algorithm. Positions that are too close to a wall are pruned. In the downsampled image mentioned in section 1.2, the algorithm prevents positions that are within 5 pixels width-wise and 9 pixels height-wise, effectively creating a bubble around the agent. If the agent does happen to get too close to a wall (which is indicated by A* not finding a path, because all positions adjacent to the agent's current position are too close to a wall), which can happen because of frame-skipping or the random moves mentioned before, the agent will then move in the direction opposite the closest wall. This bubble is necessary to ensure that the agent does not clip corners of walls, or even run into walls in general, because it is difficult to keep track of every pixel the agent occupies.

## 3.0 Random Agent

### 3.1 Behavior

This agent, for every frame, simply selects randomly from 18 possible actions. The action can either be a no-op, a move, or a fire. As such, the random agent does not take into consideration any aspects of the game.

# Results

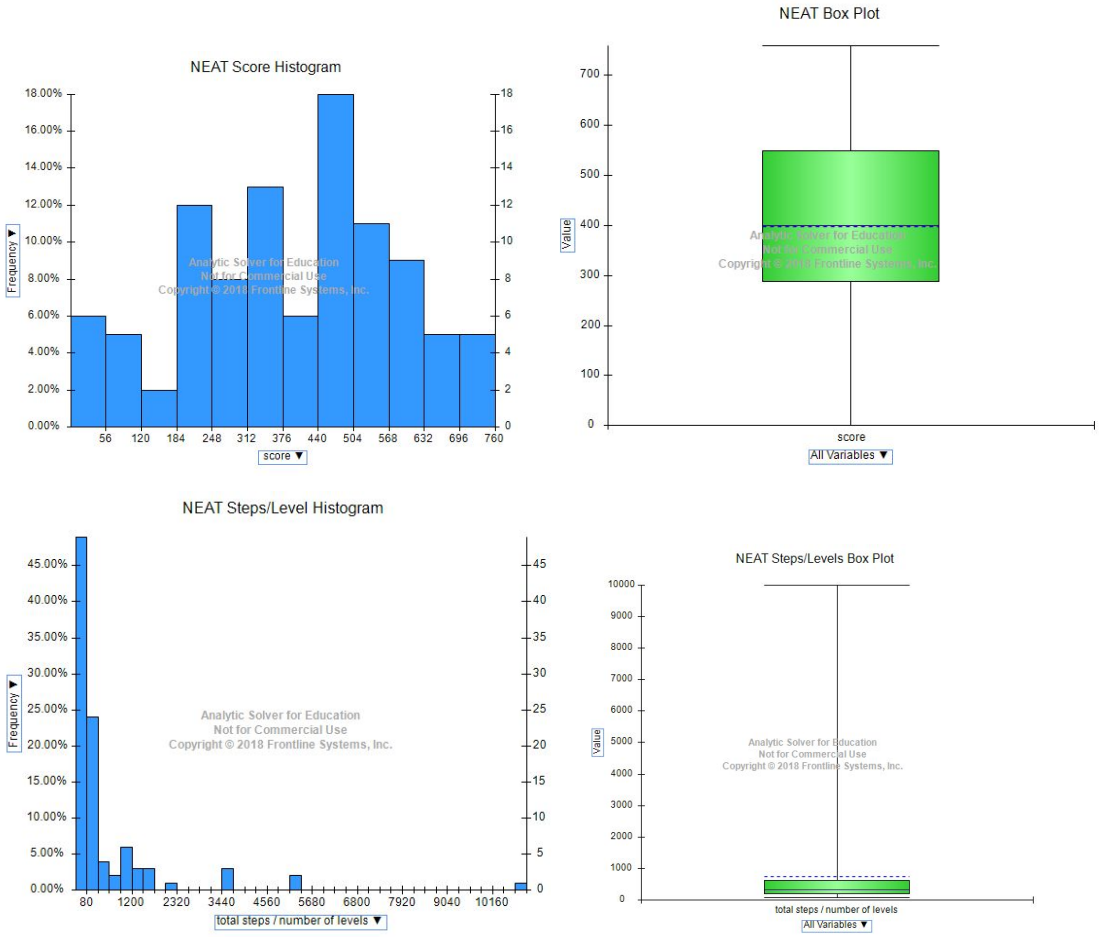
## 1.0 Neuroevolution of Augmenting Topologies (NEAT)

### 1.1 Observations

This agent definitely has made progress in terms of learning. In the first generation, it just had random connections, so it did not behave well at all. It was trained for 250 generations, and by the end it was behaving at least somewhat intelligently. Since it was trained in an environment where it would die if it went too long without moving, it rapidly moves up and down to prevent this. It has also begun to figure out when and how to fire at enemies. However, it has some major limitations. It still has a lot of trouble not running into walls, as it will often run full force at them. Its most limiting drawback is that it has not yet figured out that getting to the exit is important. For this reason, this agent never makes it out of the first room. This agent

could perform much better given a longer training time, but for the training that it did, it was able to make clear progress.

## 1.2 Statistics




For NEAT, the score had a mean of 398.8. It had a fairly wide spread, ranging from 0 to 760. The mean steps/level was 735.16 steps. The steps/level of NEAT had an average of 735.16 which is very high. This is because it did not die immediately but could not advance in levels.

# 2.0 Reflex Agent

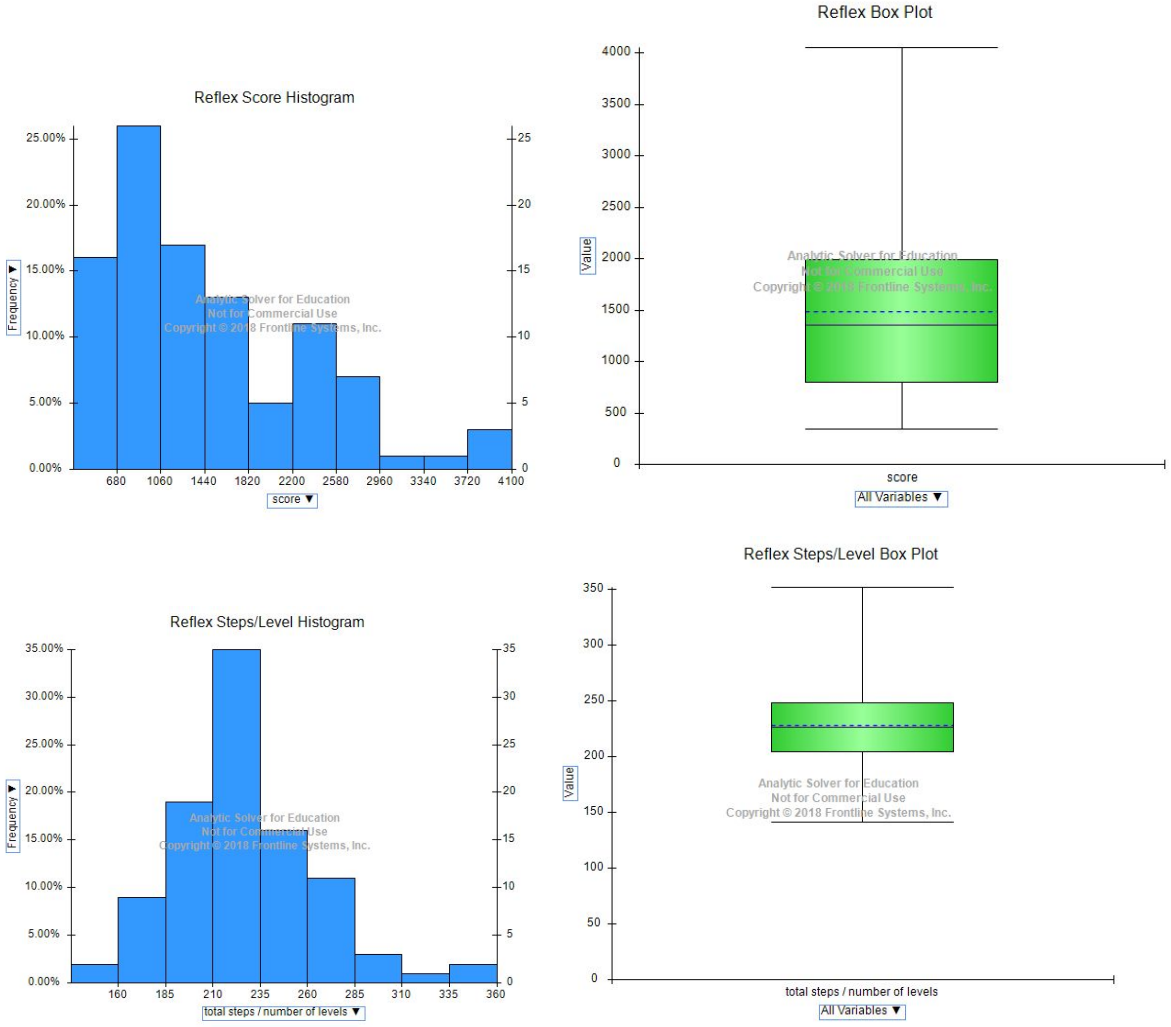
## 2.1 Observations

The agent seems to perform well, but limitations in how it observes its surroundings are apparent. First, the agent is limited to its line of sight, which originates from a single pixel and expands in 16 directions. This makes the agent miss threats that may just be out of its line of sight. In addition, the line of sight origin is not located exactly from where the agent fires from, which can cause near misses.

One of the most apparent limitations is that the agent can not sense and think that it needs to get out of the way of an incoming enemy bullet. Combined with the difficulties with getting the vision to line up with threats, this is the most common cause of death for the agent.

Since the agent looks for enemies with its vision originating from the approximate location of its arm (since it can only shoot out of there), the agent will often get hit in the head or the feet when coming around a corner.

## 2.2 Statistics



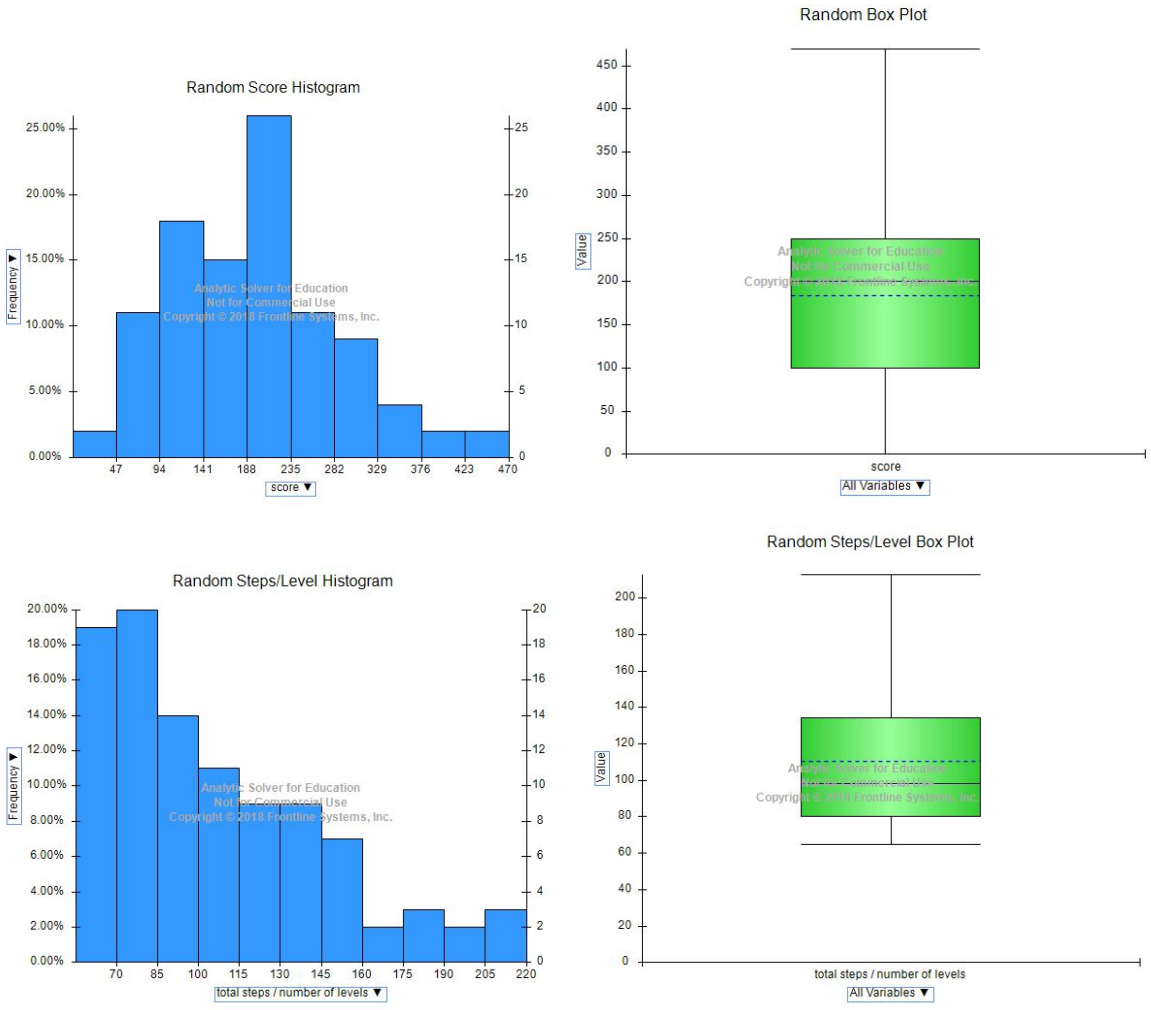

The score of the reflex agent was by far the best. It had a mean of 1491.2, ranging from 350 to 4060. This is a huge range, but it was often significantly higher than any of the other two agents, with its minimum score only barely lower than the average of the NEAT agent and the maximum of the random agent. The steps/level of the reflex agent had an average of 228.56 which shows that it got to many levels and did it fairly efficiently.

# 3.0 Random Agent

## 3.1 Observations

Because this agent is random, the agent has a wide, but generally poor, range of performances. Its most common cause of death is hitting a wall, as it usually does this before an enemy agent gets an opportunity to kill it. As such, the random agent generally does not make it past 3 levels.

## 3.2 Statistics




The steps/level of the random agent had an average of 109. This is much lower than any of the other two agents because it always died quickly.

## 4.0 Comparisons

### 4.1 ANOVA Test

What follows are the results of single-factor ANOVA test that compares each of the three agents with their respective scores for each of the 100 trials:

| *Source of Variation* | *SS* | *df* | *MS* | *F* | *P-value* | *F crit* |
|---|---|---|---|---|---|---|
| Between Groups | 98305400.67 | 2 | 49152700.33 | 195.5849536 | 0 | 3.026153369 |
| Within Groups | 74639443 | 297 | 251311.2559 | | | |
| | | | | | | |
| Total | 172944843.7 | 299 | 195.5849536 | | | |

This test got a p-value of 0, and an F-ratio of 195.585. Since the p-value is much higher than 0.05, and the f-ratio is significantly higher than the f-critical value, the null hypothesis can confidently be rejected. This means that there was statistical significance in the difference between the effect of the agent on the score. It can be said that there is a significant difference between the three tests, which means we can conclude that at least one agent is significantly different than a random agent.

# Conclusion

Overall, we were able to create two agents that did markedly better than a random agent. The reflex agent was by far the best, outpreforming the others hugely in terms of score. However, it sacrificed this increase in score for a decrease in time as it took much longer to run than NEAT or a random agent. NEAT was able to be statistically significantly better when compared with a random agent, but it also came nowhere close to the scores achieved by the reflex agent. However, it was still very fast to run as it only had to analyze the board and then get its move from the neural network instead of running A* like the reflex agent. Overall, we would like to see what results would occur if we had more training time for NEAT, and more optimizations for the reflex agent. Given these, both of these agents could perform much better.

,