Evolving robots to play dodgeball

Uriel Mandujano and Daniel Redelmeier

Abstract

In nearly all videogames, creating smart and complex artificial agents helps ensure an enjoyable and challenging player experience. Using a dodgeball-inspired simulation, we attempt to train a population of robots to develop effective individual strategies against hard-coded opponents. Every evolving robot is controlled by a feedforward artificial neural network, and has a fitness function based on its hits and deaths. We evolved the robots using both standard and real-time NEAT against several teams. We hypothesized that interesting strategies would develop using both evolutionary algorithms, and fitness would increase in each trial. Initial experiments using rtNEAT did not increase fitness substantially, and after several thousand time steps the robots still exhibited mostly random movement. One exception was a defensive strategy against randomly moving enemies where individuals would specifically avoid the area near the center line. Subsequent experiments using the NEAT algorithm were more successful both visually and quantitatively: average fitness improved, and complex tactics appeared to develop in some trials, such as hiding behind the obstacle. Further research could improve our rtNEAT algorithm to match the relative effectiveness of NEAT, or use competitive coevolution to remove the need for hard-coded opponents.

1 Introduction

Dodgeball is a children's game played indoors or outside. A rectangular playing area is divided in two, with each half belonging to one team. Players on each team try to hit each other with one of several balls without crossing the center of the arena or going beyond the boundaries. Several variations exist, but in general the object of the game is to hit an opponent with a ball without getting hit yourself. In our experiments, we used two different evolutionary algorithms (NEAT and rtNEAT) to train one team to perform better against several hard-coded opponent teams. Fitness was mostly a function of hits and deaths, so various strategies – such as staying close to the obstacle to get a safe hit – were only rewarded implicitly. We used the C++ implementations of NEAT and rtNEAT, available on Kenneth O. Stanley's website [4].

1.1 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) is "a method for evolving artificial neural networks with a genetic algorithm," developed by Kenneth O. Stanley in 2002 [4]. It replicates the natural evolution process to improve the fitness of a population of artificial agents over time. This is accomplished by altering the topology of the neural networks that control each agent based on the best-performing individuals in the previous generation. NEAT's main advantages over other neuroevolutionary machine learning methods are its built-in ability to complexify a network in a manner that maintains previous learning, while using speciation to protect new innovations that may take time to optimize. Finally, NEAT implements random mutations (i.e. changing the weight of a connection or adding or removing a node) to ensure that the population is constantly evolving. Consequently, NEAT is often superior to backpropagation since it avoids the problem of local optima in the topology. Previous experiments have used NEAT to evolve two separate populations against each other in games, modeling competitive coevolution. A study by Stanley and Miikulainen used a game similar to tag called Robot Duel as the platform for NEAT, and describe the process in more detail [6]. Another study by Mandujano and Redelmeier used Capture the Flag as the platform instead [2].

1.2 rtNEAT

Real-time (rt) NEAT is an evolutionary algorithm that adds congruity to the original NEAT engine. Whereas NEAT makes changes to the entire population at each generation (effectively resetting all agents being evolved), rtNEAT removes the worst individual every few time steps, replacing it with a mutated brain from a high-performing species. And instead of each individual having a specific fitness, agents are given a period to develop fitness, which is then updated at each time step and averaged over its lifespan. In addition, a dynamic compatibility threshold avoids the problem of one species becoming too dominant. rtNEAT was first described by Stanley, Bryant, and Miikkulainen, who used it as the basis for experiments involving the NeuroEvolving Robotic Operatives (NERO) videogame [5].

We were particularly interested in previous work that applied NEAT or a variation thereof to games in the real-time strategy (RTS) games [1]. This genre is defined by quick decision-making and reactionary play, where a player must adapt their strategy to small changes in enemy behavior. RTS games are quite common, and our dodgeball simulation is one example. Jang, Yoon, and Cho successfully used NEAT to train agents in an RTS game called Conqueror. One problem they encountered, however, was that NEAT was unable to efficiently improve fitness in networks with many inputs. We hoped to mitigate this problem in our own study by limiting the inputs to each robot to only what it needed to evolve.

Olesen, Yannakakis, and Hallam used both NEAT and rtNEAT to train AI in another RTS called Globulation 2, and rtNEAT was used to adapt the bots in real-time to compete against a specific human opponent [3]. They found that both methods were effective at improving the AI, which is particularly impressive given humans' limited attention spans. Both NEAT and rtNEAT had limitations however, especially when applied to other RTS games such as our dodgeball simulation. And instead of building up the AI from scratch, existing player-developed controllers were used as a starting point, biasing the results to some extent. One of their critiques of rtNEAT was that "it is probably better suited to games embedding more interaction between the player and the opponents, such as first person shooters, fighting games or 2D action games." Cognizant of these potential setbacks, in our experiments we attempted to replacate the success of Olesen et al. using both versions of NEAT to efficiently train a robot population, with the caveat that hard-coded opponents are used instead of human players.

2 Experiments

Our experiment setup consisted of a 500-by-1025 unit arena coded using the Simple DirectMedia Layer library. A center line splits the arena into two squares representing each team's side of the field. The robots are free to move anywhere in their region, except for over two rotationally

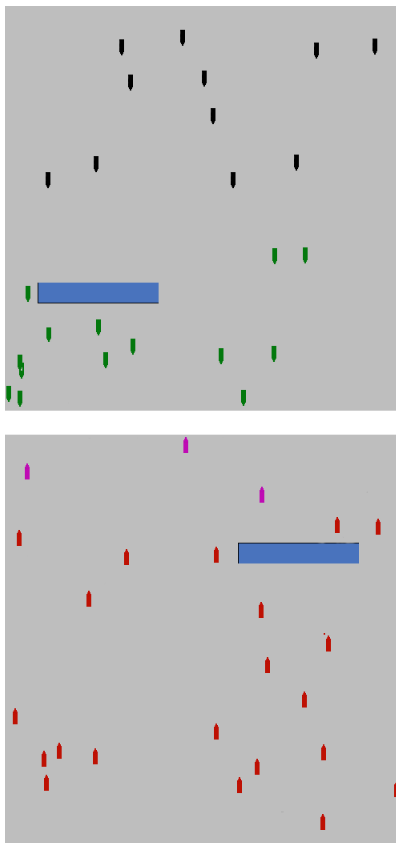


Figure 1: This picture shows the dodgeball simulator after a sample trial has just begun. The evolving population is colored black, and the individuals with targets are colored green. The hard-coded population is colored red. The pink bots are those being targeted by at least one evolving robot. The blue rectangles are the obstacles.

symmetric obstacles located near the center line. In addition, all robots had a triangular target range in front of them. The target range was used for robots to track enemies within 300 units of them in the forward y direction and up to 150 units in either x direction, scaling with increasing y distance. The evolving robot team (population size of 25) occupies the top half, and its hard-coded opponent team (also of population 25) occupies the bottom half (see Fig. 1). Furthermore, the robots had specific x and y directions that determined movement on next time step. The rules of the game were simple:

- If a robot had at least one enemy robot within its target range, it would automatically lock on to the closest target
- If a robot stayed locked on to the same target for seven time steps, the targetting robot scored a hit, except if it was hit by an enemy robot during this time
- A robot scored a death if it was hit by an enemy robot, and neither hits nor deaths reset a robot's position

- All robots could move anywhere on their side of the field, although the obstacles were impassable and blocked a robot's target range

We trained the robots against three different hard-coded opponent teams using both NEAT and rtNEAT. The first hard-coded opponent team placed every robot in a random position, which remained stationary throughout the trial. The second team was randomly created as well, but every robot could also move randomly. The third opponent was much harder, and involved coordinated movement counter-clockwise around the field (see Fig. 2). All opponents targeted a robot whenever possible. We ran four trials for each hard-coded opponent and evolutionary method, for 24 experiments total.

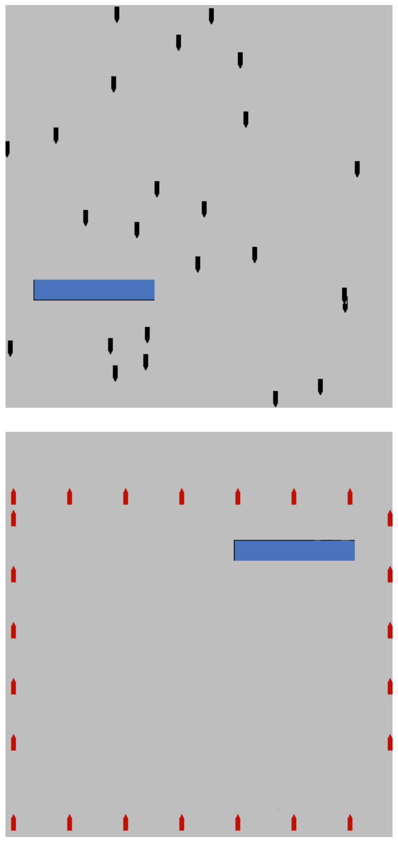


Figure 2: This picture shows the initial setup for the clockwise-rotating enemies. Unfortunately, none of the trials gave interesting conclusions, so it seems that this team was too difficult for our algorithm to learn.

The evolving robots acted independently of their teammates. They were controlled by artificial neural networks with 11 inputs and 2 outputs, as shown in Table 1. The robots had an associated fitness, primarily determined as a positive function of its hits and a smaller negative function of its deaths. We added an explicit reward for staying locked on to a target that increased with consecutive time steps to incentivize scoring hits. The details of our fitness function for our NEAT experiments are shown below. The fitness function was the same for the rtNEAT trials, except that

Input nodes	**Output nodes**
Robot's x position	Robot's x direction
Robot's y position	Robot's y direction
Robot's y distance from the center line	
Robot's x direction	
Robot's y direction	
Number of enemies targeting a robot	
Boolean (1 or 0) representing whether or not robot has a target	
Relative x distance of a robot's target (0 if robot has no target)	
Relative y distance of a robot's target (0 if robot has no target)	
Relative x direction of a robot's target (0 if robot has no target)	
Relative y direction of a robot's target (0 if robot has no target)	

Table 1: The inputs and outputs to the ANN controlling the evolving robots in all experiments.

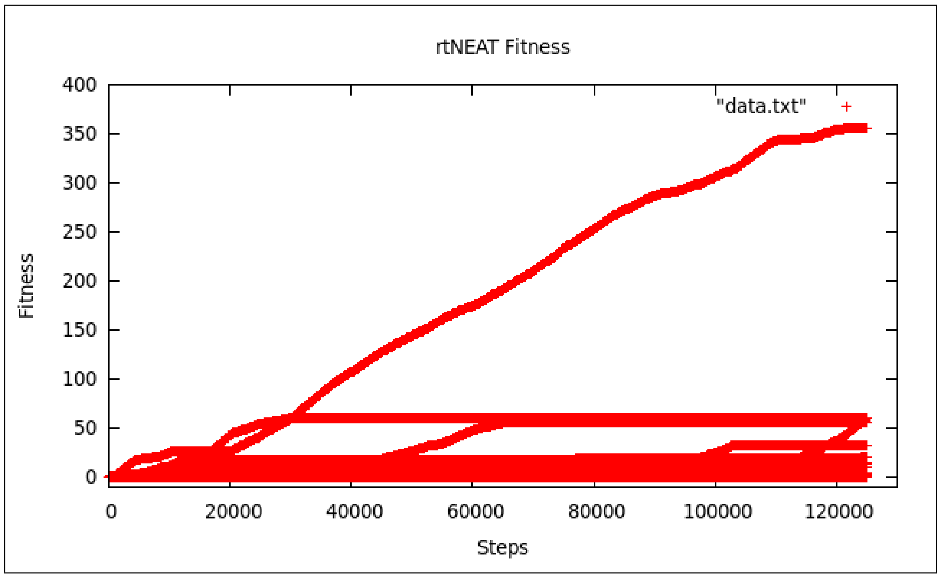
rtNEAT fitness was averaged out over an agent's lifespan while NEAT fitness was reset at every generation.

```
if fitness < 0:
   fitness = 0

if a robot escapes an enemy's target range before being hit:
   fitness += 5

if a robot locks on to an enemy:
   for every time step it stays locked on:
      fitness += 2^(time steps locked on)
   if the robot scores a hit:
      fitness += 150

if a robot is hit by an enemy:
   fitness -= 30
```

3 Results

In our rtNEAT experiments, we ran the trials for 125,000 time steps, updating fitness and replacing low-performing individuals every 500 steps (see Fig. 3). For 11 of the 12 trials, fitness did not increase significantly, and visually the robots did not noticeably change their behavior in a meaningful way. One successful run was our second trial using the randomly moving hard-coded population. The robots gained higher comparative fitness values, and we saw that they acted defensively. This led us to believe that our fitness reduction on being hit by an enemy may have been higher than necessary.

Our results for the NEAT experiments were more fruitful. We saw fitness rise to higher levels (see Fig. 4), and defensive strategies developed in most trials. As with the successful rtNEAT trial, two of the best runs in our NEAT experiments occured when the robots were trained against



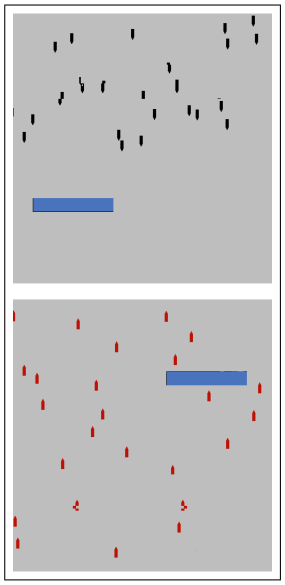


Figure 3: The picture on the left shows the progression of average adjusted fitness for all 12 rtNEAT trials. We observed that almost all trials did not develop any observable strategies or increase fitness by much. Trial 2 using our randomly moving opponent reached a fitness of almost 400, which corresponds to a relatively defensive tactic where the robots avoided the center. This is shown visually on the right picture. We believe that the robots developed this strategy by staying close to or at the maximum target range of the hard-coded bots as they approached the center line. Since targets were acquired after a robot moved, the evolving robots would get a “head start” by targeting the enemy robots before they became targets as well. This would allow the evolving robots to score a hit first and increase fitness.

the randomly-moving team. In both of these tests, we saw that the robots developed defensive strategies that used the obstacles to target enemies more safely (Fig. 5). Unfortunately, the robots still underperformed against the coordinated rotationally-moving enemy, and none of the four runs yielded high average fitness or interesting visual results.

4 Discussion

Our results add further evidence supporting the effectiveness of using NEAT and rtNEAT to train AI in semi-predictable situations, provided there is an appropriate fitness function used. In our dodgeball simulation, rtNEAT was only useful in one of the twelve trials. In this trial, the robots learned a defensive tactic by positioning themselves away from the center, and targetting any random bot that came too close. Our NEAT trials were more conclusive and gave higher fitness values overall, and demonstrated that the robots could evolve a more complicated strategy that made use of the obstacle. Further testing could manipulate the NEAT and rtNEAT settings (such as the probability of adding or removing a node), which were kept at their default values in all of our experiments.

Overall, we believe that our trials were successful for NEAT, and not entirely unsuccessful for

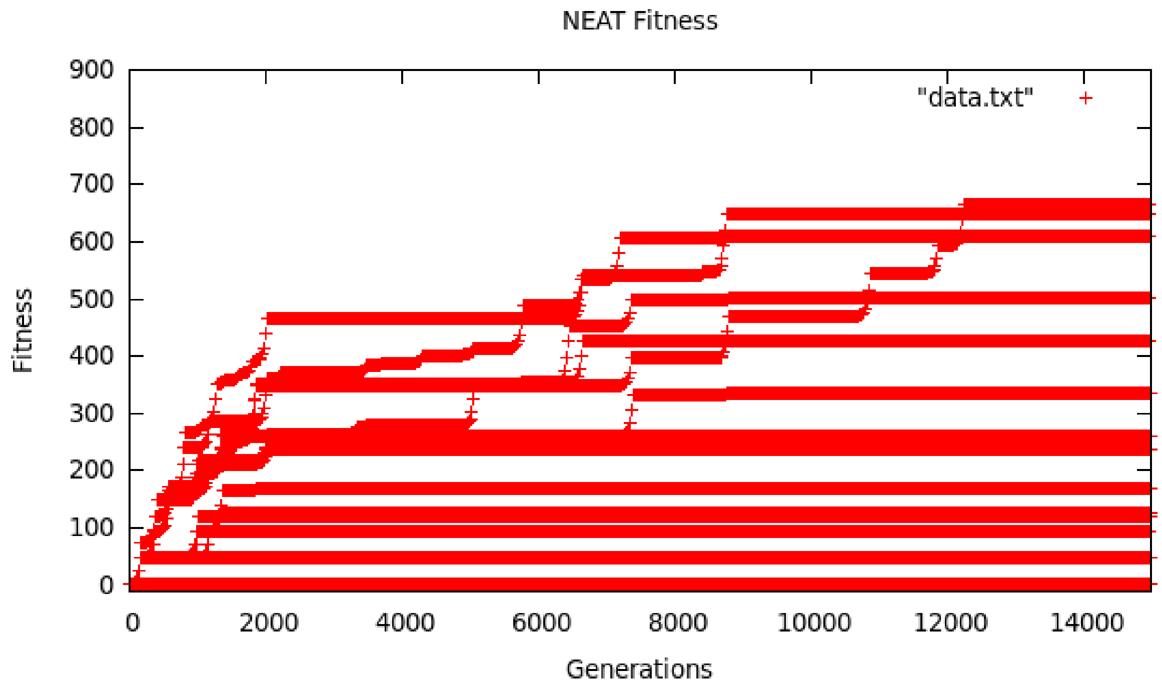


Figure 4: This graph shows fitness progression for all 12 NEAT trial runs. Although many runs plateaued in fitness after a certain generation, they still outperformed rtNEAT on average in efficiency and fitness (although it is difficult to compare the two fitness algorithms since they are calculated differently in NEAT and rtNEAT). Akin to rtNEAT, the highest fitness levels were attained in trials against the randomly moving opponent team. And despite the success, fitness remained low during trials against the rotationally-moving enemy.

rtNEAT. If we are able to run more generations and develop better topologies, we may see the rise of robots that can reasonably compete against a human or adaptive controller as opposed to one that is hard-coded, developing unique, innovative strategies in the process. Future research could make use of competitive coevolution to evolve the robots against continuously changing oponents, as in [6] and [2]. Furthermore, we hope to improve the rtNEAT method specifically to match the performance standard acquired in most of our NEAT trials. Finally, we hope to use similar methods to evaluate the applications of these two evolutionary algorithms in other RTS games.

5 Acknowledgements

We would like to thank Lisa Meeden for providing guidance on the project, particularly for our target-finding algorithm. We would also like to thank Kenneth O. Stanley and Peter Chervenski for providing more information about the rtNEAT package. Finally, thanks to Teo Gelles and Mario Sanchez for helping us implement rtNEAT successfully and debug our simulation.

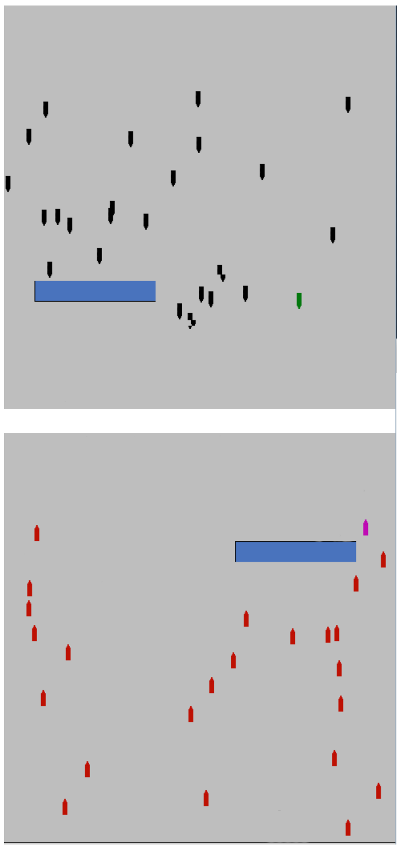


Figure 5: This picture shows the game state using the 13000th generation chromo in our best-performing NEAT trial, which competed against the randomly-moving opponent. We saw high fitness values at this stage, and noticed visually that the robots evolved a sophisticated strategy by staying next to or behind the obstacle where it was safest.

References

[1] Su-Hyung Jang, Jong-Won Yoon, and Sung-Bae Cho. *Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm*, pages 75–79. IEEE Press, 2009.

[2] Uriel Mandujano and Daniel Redelmeier. Evolving robots to play capture the flag. *Swarthmore College Department of Computer Science*, 2014.

[3] Jacob Kaae Olesen, Georgios N Yannakakis, and John Hallam. *Real-time challenge balance in an RTS game using rtNEAT*, pages 87–94. IEEE Press, 2008.

[4] Kenneth O. Stanley. The neuroevolution of augmenting topologies (neat) users page. `http://www.cs.ucf.edu/~kstanley/neat.html`, 2013.

[5] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. *Evolving neural network agents in the NERO video game*. IEEE Press, 2005.

[6] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21, 2004.