Evolving Combat Robots

Dylan Katz, Greg Rawson, Jessi Velasco

May 12, 2012

Abstract

The ultimate goal of this project is to apply a NEAT controller to a simulated combat environment. Our immediate goal became to evolve a single robot and have it battle a hard-coded enemy and then gradually increase the complexity of the task by adding multiple enemies and robots or increasing the complexity of the environment until our ultimate goal was accomplished. The combat environment was a simple 6x6 grid world but only 4x4 can be occupied because the outside was all walls. The battles were turn based. A lot of time was spent trying to make sure that the single robot was actually evolving and were therefore unable to create the heterogeneous swarms as originally planned. However, after much difficulty we were able to successfully evolve a single robot. Overall after several trials, we managed to get a working but inelegant robot that could survive in the combat environment.

1 Introduction

Our goal for this project is to apply developmental, adaptive techniques within a simulated combat environment. We believe that combat simulations provide a unique set of challenges, and that even though adaptive systems aren't the current industry standard, they can provide a unique perspective. Many of the systems we've seen have been occasionally disappointing in their limited scope. Knowing that we couldn't do any better, we chose to reduce the realism of our simulation environment. This gave us a bit more freedom in designing our experiments since we cheated and ignored the inherent complexity of the real, physical world. We aim to demonstrate that our approach has merit, despite its compromises.

Given the time constraints of the project, we set out to create a basic but extensible simulator that would become a test bed for new features and tweaks. This simulator provides a battleground for our simulated RoboSoldier to fight in. We prioritize mental growth, happily assuming basic physical capabilities. Specifically, we want to encourage the emergence of strategies and tactics in our developmental robot. To this end, we built a simple turn-based, tile-based environment, with discrete time and space. We felt this provided a streamlined environment for our embodied agent, as well as giving us valuable control over the environment which was invaluable in the debugging stage.

The last piece of the puzzle was finding a suitable learning system to serve as the brains of our virtual protagonist. This was a fairly simple process, as we only seriously considered two different solutions. Our first thought was to use SODA and Queue-Learning. Dylan and I had had good luck with it in the past with the tic-tac-toe AI we built. We thought it would be a natural fit given our discretized environment. There is also something to be said for being able to look under the hood and easily understand how it's working. However, we were concerned with the inevitable physical memory limitations of storing a huge state table, as well as the inability to dynamically adapt to new environments. This drove us to look at neural networks and the NeuroEvolution of Augmenting Topologies (NEAT) method. NEAT is a brilliant mix of neural networks and genetic algorithms, applicable to many problems through its user-configurability and inherent adaptability. NEAT is effective at finding efficient solutions in very large search spaces. NEAT evolves neural networks with minimal human setup required, making the entire process very organic. NEAT possesses one feature that is of particular value to our goal of observable strategy and tactics - the ability to elaborate on its existing strategies, through systematic complexification of current solutions.

2 Related Work

In the paper Real-*Time Evolution of Neural Network*, the authors presented the NERO Video Game. NERO uses a modified form of NEAT - rtNEAT - that is able to evolve and adapt in real-time. This version of NEAT is used to build the NeuroEvolving Robotic Operatives (NERO). NERO is a new type of machine learning video game where the player is able to train units in real-time in order to perform difficult tasks in a virtual environment. The neural networks are complexified as the game is played, which makes it possible to have units evolve interesting sophisticating behaviors in real-time. The rtNEAT implementation was flexible and robust enough to achieve interactive learning in real-time in challenging sequential tasks [2].

In another paper, another specialized version of NEAT is presented. Known as cgNEAT (content generating NEAT) is used to evolve a game as it’s played based on player preferences.

As described in the paper, *Evolving Content in the Galactic Arms Race Video Game*, cgNEAT is used on the video game Galactic Arms Race (GAR) in which players control spaceships in order to gather weapons that are evolved by the game to fight enemies. A wide variety of weapons that are both new and have been expanded from previous weapon are discovered by the player. cgNEAT could allow for games to generate their own content yielding a more immersive, personal experience [1].

It seems natural to us that someone will incorporate both of these systems in the same virtual environment, which has the potential to revolutionize the virtual worlds used today. In *Cooperative Multi-Step Behavior in an Evolved Robot Team*, simulated robots are placed in an environment that allows encourages teamwork and societal roles to be evolved. There is a group of gathering robots as well as a group of hard-coded predatory robots. The predators can eliminate the gatherers by reducing the gatherers' health, but the reduction in health is limited by the allies that the gatherer has around. This mechanism led the gatherers to evolve several grouping and herding behaviors that were used in their final strategy for survival. While the 2 super-NEAT papers were interesting, we were in some ways more inspired from seeing the success that Ryan had with his project [3].

Overall, we didn't find many papers that seemed super similar to our experiments. We briefly looked at papers for adaptive AI systems for Civilization-style games, and game AI in general. Adaptive systems aren't that commonly used, with most things be scripted or cheating the system to add difficulty. We realized that even though our simulation isn't directly applicable to real world systems, we feel safely on the robotics/embodiment side of the scale.

3 Experiment

3.1 Combat Environment

The combat environment consisted of a 6x6 grid world but only the smaller 4x4 inside the 6x6 grid can be occupied because the outside consisted of walls. The world was small because we expected a lot of encounters would happen between the enemy and the robot, which in turn would lead to interesting learning behavior. Once we saw that there was no interesting learning behavior occurring then we would increase complexity by making the world. The enemy always started in the top left corner while the robot started in the bottom right corner. Figure 1 below shows the beginning of each battle.

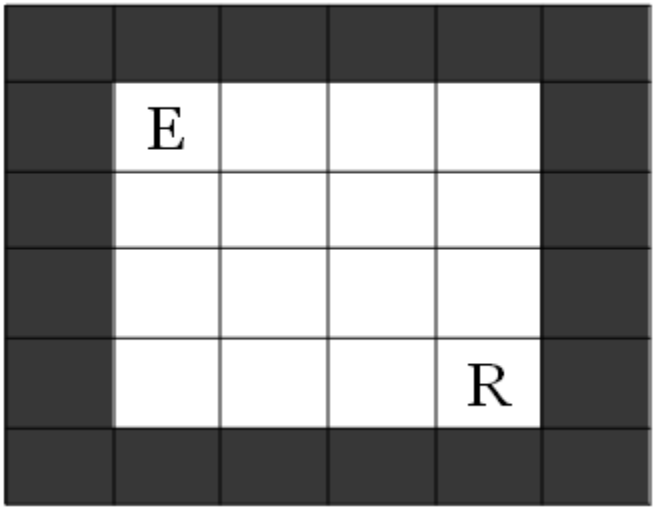

Figure 1: The grid world that served as the combat environment. The white 4x4 grid is the space that can be occupied while the colored tiles represent the walls. The robot always starts at the bottom right corner while the enemy starts in the top right corner.

Facing was taking into account in our implementation. The facing of the robot and enemy would depend on the last move made. For example if the last movement was left, then the unit would face left at the end of its turn. Facing also played a role in the field view of the units. Field of view went clockwise starting on the units left. The unit could see a blank tile, a wall, or the opponent. Figure 2 shows examples of the field of view of a robot.

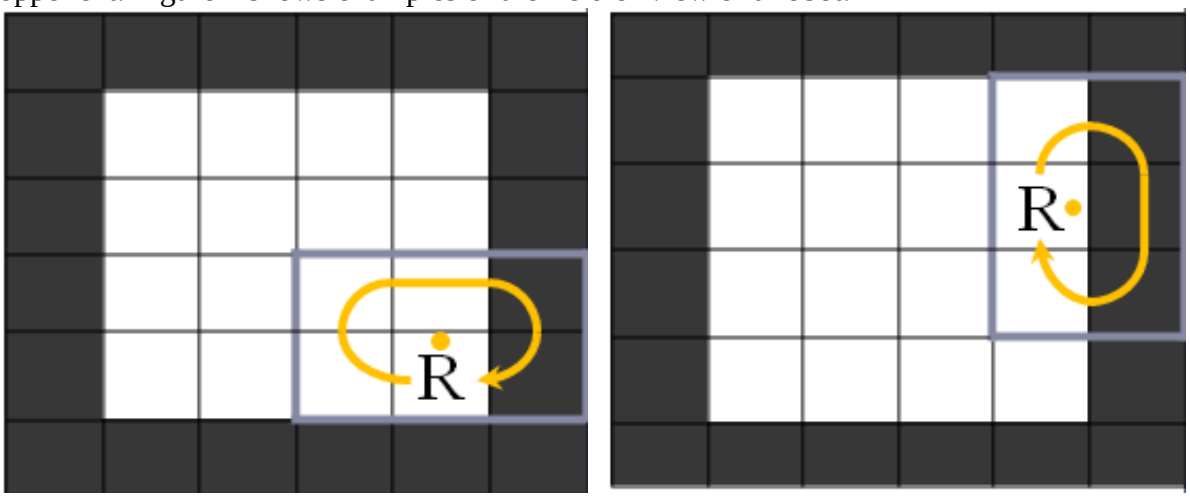

Figure 2: These are two examples of the field of view for a robot. The gray outline is what the robot can see and the dot represents the facing of the robot. In the left the robot is facing up and in the right the robot is facing left. The field of view starts clockwise from the left of the robot which depends on the facing.

The enemy was hard-coded and implemented the in the following. If it saw the enemy in its field of view, then it would shoot, otherwise it would move randomly. The enemy could only shoot if it saw the robot and it could never hit a wall. The robot was NEAT controlled and unlike the enemy, it could shoot whenever it wanted and can walk into walls. We anticipated that the robot would learn to avoid walking into walls and the proper occasion to shoot. Both started with a full health of 100 and each unit can shoot 4 times. If a unit got shot then it would lose 25 health. Collisions between the robot and the enemy can occur and each would lose 10 health. These collisions happen by chance and are not done on purpose. The battle was turn based both the robot and the enemy can choose to move up, down, right, left, or shoot. The robot always went first and a round in the battle consisted of 1 enemy turn and 1 robot turn. The battle would continue until one unit died or until 100 rounds have expired. A round would be as follows: 1) the inputs would be gathered and passed into NEAT, 2) NEAT would give outputs and the maximum output would be applied, 3) the world would be updated, 4) the robot would choose an action based on its simple implementation, 5) the world would be updated, 6) check for a winner, and repeat. Figure 4 shows a round.

3.2 NEAT

As it was mentioned above, the robot was controlled by a NEAT controller. A recurrent neural network is used in this project meaning that connections between units form a directed cycle. There were six inputs into NEAT for the robot: 5 for the field of view and 1 for health.

The inputs for the field of view where set as following: 0 is used to denote a blank tile, 0.5 is used to denote wall, and 1 is used to denote a tile occupied by an enemy. Figure 3 shows an example of the inputs that the robot would have just for a specific field of view. The robot's health was normalized so the input to NEAT would be a float between 0 and 1. The outputs for NEAT were up, down, right, left, and shoot and the maximum output was applied. It started with 0 hidden nodes, but hidden nodes were added as the robot evolved. The probability for *addconn* is 0.07 and the probability for *addnode* is 0.05.

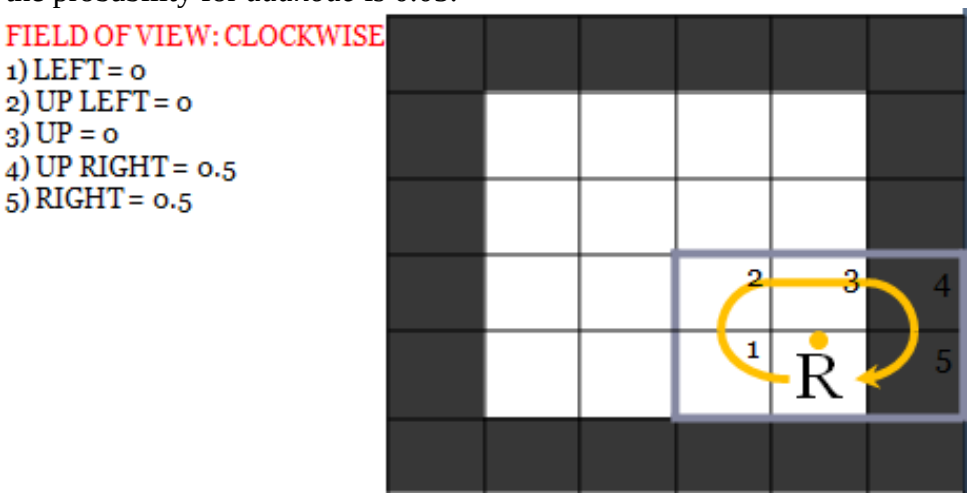

Figure 3: An example of what the inputs would be for the robot for its field of view. 0 represents a blank tile, 0.5 represents a wall, and 1 represents an opponent.

3.3 Fitness Function

The fitness function took into account 3 elements: the number of shots hit, the number of unique tiles visited, and the amount of health left. The number of shots hit rewards shooting, the number of unique tiles visited rewards exploration, and the amount of health left rewards survival. We want the robot to be to learn to shoot and kill the enemy, but it will take a while for the robot to learn how to perform shooting properly, so the fitness function could not solely depend on shooting. As a result other elements besides shooting were rewarded, but shooting was still the most rewarded. Survival was the second most rewarded leaving exploration to be the least rewarded. The equation below used to calculate the fitness is shown below:

$$F = 0.2H + 12S + 2T$$

The coefficient where chosen so that the maximum fitness would be 100. Health ranged from {0, 100}, shots ranged from {0, 4}, and tiles ranged from {0, 16}. Once the coefficients are applied, the ranges for health, shots, and tiles changed to {0, 20}, {0, 48}, and {0, 32} respectively. If the maximum of these 3 elements are added, then we get a maximum fitness of 100 as mentioned above. If important to note that a maximum fitness of 100 cannot be reached because it means that every tile must be visited and this is highly unlikely in the combat environment that we have as of now. The world is small so the enemy and robot are going to encounter before the robot has a chance to visit every tile. We found out that good fitnesses are generally in the range 57-63. This means that the robot usually has 25 health left, has fired 4 shots, and has visited 2-5 tiles.

Originally the fitness function rewarded survival the most, shooting the second most, and exploration the least. Unfortunately with this fitness function, the robot was not learning to shoot at the enemy at all. There would be instances where the enemy would be in its field of view and

the robot would not shoot at all. After several trials we didn't see any shoot be fired from the robot. Instead, the robot would end up getting killed most of time by the enemy. As a result we decided to make the robot more aggressive by rewarding shooting the most.

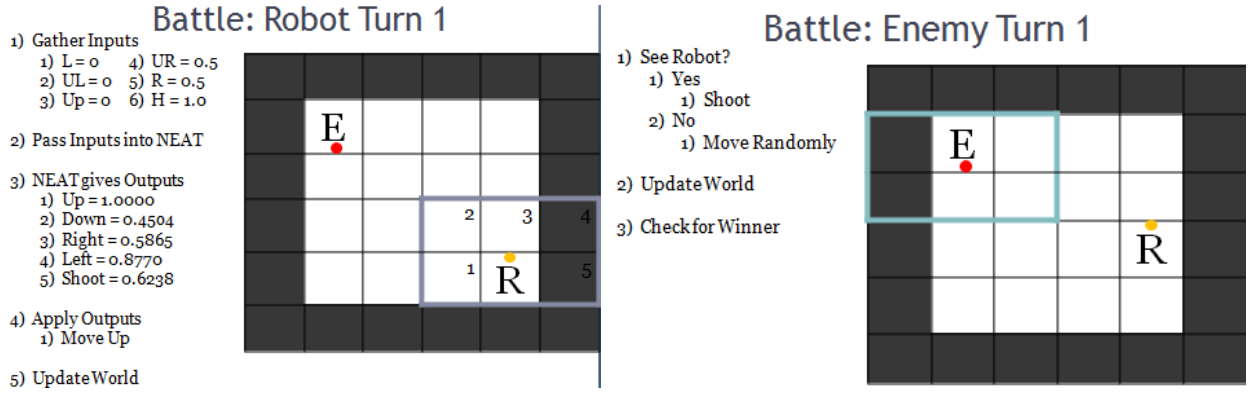

Figure 4: A round between the robot and the enemy. The inputs are gathered and passed into NEAT. NEAT gives the output and the maximum output is applied. The world is updated and the robot moves based on the implementation written. The world is updated once again and we search for a winner. This repeats until a unit dies or until 100 rounds have expired.

4 Results

Before successful results were obtained some changes were made in order to assure ourselves that the robot was indeed learning. Some of these changes included changing parameters such as the number of rounds, generation size, and population size in order to confirm that the robot was actually learning. Other changes included having the robot and enemy start at fixed locations. Originally the placement of the units on the world was random because we thought this would lead to robust behavior. Unfortunately, no learning was done by the robot so as a result we decided to simplify the environment further by having the units start at fixed location.

The following options parameters had good fitnesses: a population size of 200, a generation size of 200, a species size of 10, 10 battles, and 100 rounds in each battle. Figure 5 below shows the population's average and best fitness.

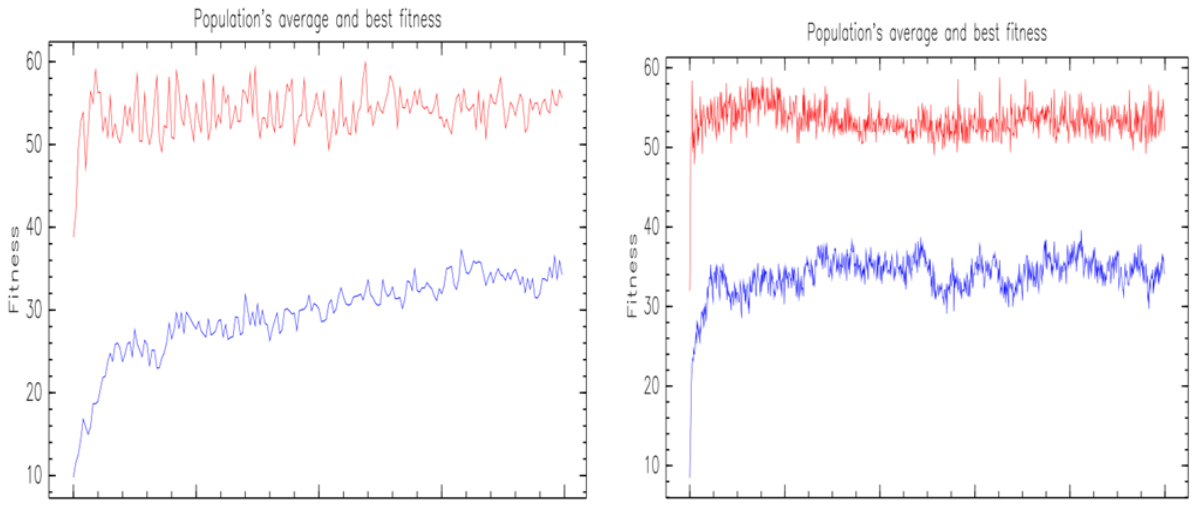

Figure 5: The population's average and best fitness are displayed. The left graph has a population size of 200 and the right has a population size of 1000.

As we can see from the left graph of Figure 5 has a population size of 200 and we see the fitness starts to increase. As the population size increases we expect that the average fitness is no longer going to increase and level of at some point. This can be seen in the right graph of Figure 5 which has a population size of 1000. Also in the right graph we can barely see the increase that occurs in the left graph. Most of the graphs had this general outline.

The following shows a run through of one of the successful battles that occurred. Figure 6 shows the start of the round and it was stated above the robot starts bottom right corner and the enemy starts at the top right corner.

```
######
#E___#
#____#
#____#
#___R#
######

Robot Health 100
Robot Facing up
Robot Field of View blank blank blank wall wall
Enemy Health 100
Enemy Facing down
Enemy Field of View blank blank blank wall wall
```

Figure 6: Each battle begins the same with the robot in the bottom right corner and the enemy in the top left corner. Information pertaining to the robot and the enemy are displayed.

We display the initial health of the units as well as their facing and field of view. The robot decides to move up and the enemy decides to move and these movements are shown in Figure 7.

```
######
#____#
#E___#
#___R#
#____#
######

Robot Health 100
Robot Facing up
Robot Field of View blank blank blank wall wall
Enemy Health 100
Enemy Facing down
Enemy Field of View blank blank blank wall wall
```

Figure 7: The results after 1 round. 1 round consists of a robot turn and an enemy turn.

For next several of turns nothing to interesting happens. The robot and enemy keep moving around the environment. Figure 8 shows the movements that the enemy and the robot go through until the eventually they see each other and more interesting results happen. In one occasion, the 7 diagram in Figure 8, the robot moves into wall and loses 10 health.

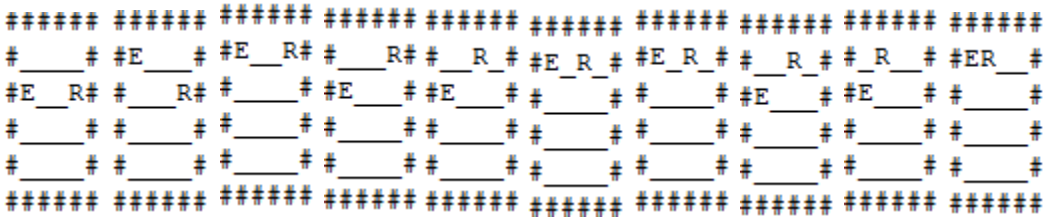

Figure 8: The movements that the robot and the enemy go through until they come into each other's field of view.

As seen in the last diagram in Figure 8 the units come into the view of each other. It's the robot's turn the robot decides to shoot the enemy. During the enemy's turn the robot also decides to shoot and a shootout between the two occurs. Whenever a shootout occurs, the unit to shoot first is usually the one to win. Figure 9 shows information pertaining to the health and we can indeed confirm that the shooting action occurred with the 25 health decrease that the enemy experiences.

```
Robot Health 90
Robot Facing left
Robot Field of View blank blank enemy wall wall
ACTION shoot

######
#ER__#
#____#
#____#
#____#
######

Robot Health 90
Robot Facing left
Robot Field of View blank blank enemy wall wall
Enemy Health 75
Enemy Facing up
Enemy Field of View wall wall wall wall robot

Enemy Health 75
Enemy Facing up
Enemy Field of View wall wall wall wall robot
ACTION shoot
```

Figure 9: The robot is the first to fire and the enemy retaliates and thus a shootout occurs.

Figure 10 shows the end result of the battle and information pertaining to the battle are displayed such as the remaining health, shots fired, tiles visited, and fitness for the robot.

```
######
#ER__#
#____#
#____#
#____#
######

Robot Health 15
Robot Facing left
Robot Field of View blank blank enemy wall wall
Enemy Health 25
Enemy Facing up
Enemy Field of View wall wall wall wall robot
Bimbo Died!
Health: 15, Shots: 4, Tiles Visited: 6, Fitness: 63
```

Figure 10: The robot wins the shootout. Bimbo is the name of the robot. Information pertaining to the battle is displayed.

5 Conclusions and Future Work

Overall, we're satisfied with what we've been able to achieve. It certainly wasn't the impressive tactical warrior that we had hoped for, but I think our expectations were too high to begin with. We were sort of counting on miracles from NEAT, but didn't realize the amount of work, experimentation, and luck that goes into finding superb solutions. We momentarily forgot that as wonderful as NEAT is, it's still a search algorithm. I wish we could say we have mastered NEAT, but I don't think anyone can truthfully claim that.

Since we presented our results, we've made further progress with our simulator. We changed the shooting system to allow the robot to shoot whenever, and implemented an ammunition system to go along with this. At first we saw a drop in average and best fitness, but after fiddling with things we saw big improvements by reducing our weight_mutation_power from a high 4.5, to the normal 1.5. Reducing the species size from 20 to 10 gave us another boost, and brought us back to our previous fitness levels. Next, we increased the field of view to much more generous 14 tiles, made the map much bigger - 10x10 , and decoupled our facing/turning system from our movement system. As of now, everything works but the real challenge is finding the optimal configuration for NEAT. A few areas that seem promising our minimizing our output nodes by mapping many onto one. Something else we want to try is to explore other activation functions. Using tangent, all the outputs are likely 1.0 or -1.0. This will require a bit of research as well as the good old trial and error. When we added the most recent features, we made things more modular and parameterized.

6 References

[1] Hastings, E., Guha, R., and Stanley, K.O. 2009. Evolving Content in the Galactic Arms Race Video Game. In Proc. of the IEEE Symposium on Computational Intelligence and Games (CIG ’09). Milano, Italy. September 7 – 10, 2009. Web.

[2] Stanley, K. O., Bryant, B., and Miikkulainen, R. 2005. Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation, 9:653–668. Web.

[3] Walker, Ryan. 2009. Cooperative Multi-Step Behavior in an Evolved Robot Team. Web.