EECS 545, Machine Learning
Winter 2012, University of Michigan, Ann Arbor *Date: March 8, 2012*

Project Progress Report

TEAM MEMBERS:
Keegan R. Kinkade, @kinkadek
Pedro d'Aquino, @pdaquino
Shiva Ghose, @gshiva

Introduction

Initially released in 2001 with the aim of assisting in learning object oriented programming, Robocode is an open source tank battling simulator implemented in Java. Each Robocode agent is designed with the ability to control a robotic tank, including differentially-driven movement, a 360° rotating turret, and an enemy scanning radar, in order to attempt to destroy other tanks with similar features but opposing strategies. Designers are capable of downloading other agent byte code in order to test their implementation, and there also exists leagues in which Robocode tournaments are held. While the environment has made creating a simple working agent capable of moving, targeting, and shooting, perfecting agents to do well against multiple opponent strategies has proven to be difficult. Many of the top ranked agents incorporate sophisticated statistical analysis techniques in order to optimize their agents. In addition to such techniques, research has been done to optimize agents using both neural networks as well as genetic programming.

Statement of the Problem

The purpose of this project is to incorporate machine learning algorithms in a Robocode agent in order to optimize the agent's ability to stay alive during battles. Specifically, we wish to incorporate machine learning in two areas: targeting and evasion. With respect to evasion, we intend on using reinforcment learning in combination with genetic algorithms to optimize multiple missile evasion strategies and employ them in such a manner as to reduce the likelihood of being struck by an opponent's missile. Furthermore, we wish to use modeling machine learning algorithms in order to predict how an opponent will react upon learning that our agent has fired, thus allowing our agent to take advantage of the ability to predict opponent's behavior.

Proposed Approach

We aim to handle this problem by building a tank A.I. that consists of two separate sub–agents, working in tandem to maximize the overall performance measure. The first sub–agent will handle the movement aspects of the tank while the second sub–agent will be tasked with modeling the opponent.

Each sub–agent will have a variety of strategies, each of which will be optimized against the opponent. The effectiveness of the strategies will also be evaluated in real time to determine which strategy or set of strategies play out the best. We propose to do this using a combination of machine learning algorithms and genetic programming methodologies. Each individual strategy will be optimized against an opponent using genetic programming methods. During the optimization period, the agent would only be allowed to use a given strategy and the various individual parameters of the strategy would then be fine–tuned over the course of the training rounds. After this the agent is allowed to use all the trained strategies against the opponent. Reinforcement learning would then be employed to determine the agents preference of each strategy as it plays against the opponent.

Movement Strategy

There are two main movement strategies that need to be considered:

- General movement strategy
- Evasive movement strategy

Each strategy needs to be used differently and the the movement agent will need to learn when and how to employ them.

General Movement Strategy

In general the probability of getting hit by incoming fire is inversely proportional to how close the observer is to the firing point. We want our agent to move closer to the enemy when it has higher health to maximize the probability of hitting the enemy and stay further back when it is lower on health to better dodge incoming fire. This can be modeled as series of attracting and repelling force interactions. Instead of hard–coding the values of the parameters, we will use genetic programming in a gradient-descent–like form to find the optimal values that our agent should use against the opponent.

Evasive Movement Strategy

Robocode allows you to figure out if the enemy agent has fired a shot, but it does not tell you anything about where the bullet is heading. Hence we need to use a combination of maneuvering strategies and statistical methods to figure out probabilistic bullet trajectories. Again, each of the parameters of each strategy will be tuned similar to the general movement strategy parameters.

Offensive Strategy

The offensive strategy deals with modeling where the opponent will be if the agent were to fire at this point of time. A simple movement vector extrapolation will often fail even against all but the most rudimentary opponents as they will also have bullet avoidance techniques. Hence a more sophisticated prediction scheme is required.

We aim to build up a probabilistic distribution of the the opponents actions and reactions over time to model the opponent and use that to get higher scoring firing solutions.

Additionally, no matter how random an opponent's movement might be, as the opponent approaches an edge or a corner of the battle field, the number of movement options become limited. We will need to create a method that can exploit this in parallel with our regular targeting scheme to get even higher scoring firing solutions.

Evaluation Methodology

In order to measure the agent's performance, we require a performance measure. The agent's health is a good place to start, though that alone will have some drawbacks.

In Robocode, in order to fire, an agent must use some of its own health. If the agent's health were it only performance measure, it is foreseeable that an agent will quickly get stuck in a local minimum by choosing not to fire at the enemy at all and instead try to survive by simply dodging the enemy's incoming fire. While this might work initially, as the agent will be drawn closer and closer to the enemy as its opponent loses health, it is easy to imagine that the agent will still get considerable damage.

Hence we need to devise a reward scheme that rewards the agent for effectively attacking the enemy, effectively dodging incoming fire and staying alive. These rewards will probably be set manually, although if time permits, these rewards can also be optimized using machine learning techniques.

Review of Related Work

We have found some articles describing genetic programming approaches to Robocode, but none that combines it with reinforcement learning. Eisenstein, in 2003, was the first to use genetic programming in this context [1]. He found that, while he was able to beat some hand-coded adversaries, his robots had a hard time learning how to target, and were therefore more likely to try to ram their opponents.

In [3], the authors build upon Eisenstein's work and use genetic programming to evolve tank strategies for a robot in the HaikuBot category (which allows robots whose code is no longer than 4 lines). They evolved a population of 256 robots over approximately 400 generations. Their robot was ranked 3rd in the HaikuBot category.

In one of the very few academic mentions of Robocode outside of the artificial intelligence field, Kobayashi et al. describe a targeting strategy that was only mildly successful [2]

References

[1] J. Eisenstein, "Gp-robocode: Evolving robocode tank fighters," *Technical Report AIM-2003-023, AI Lab, Massachusetts Institute Of Technology*, 2003.

[2] K. Kobayashi, Y. Uchida, and K. Watanabe, "A study of battle strategy for the robocode," in *SICE 2003 Annual Conference*, vol. 3, aug. 2003, pp. 3373 –3376 Vol.3.

[3] Y. Shichel, E. Ziserman, and M. Sipper, "Gp-robocode: Using genetic programming to evolve robocode players," in *Genetic Programming*, ser. Lecture Notes in Computer Science, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds. Springer Berlin / Heidelberg, 2005, vol. 3447, pp. 143–143. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31989-4_13