Improving Opponent Intelligence through Machine Learning

Pieter Spronck   Ida Sprinkhuizen-Kuyper   Eric Postma

Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht

Abstract

Artificially intelligent opponents in virtual world computer games are almost exclusively controlled by manually-designed scripts. With increasing game complexity, the scripts tend to become quite complex too. As a consequence they often contain "holes" that can be exploited by the human player. The research question addressed in this paper reads: How can machine learning be used to improve the quality of opponent intelligence in computer games? We study the off-line application of evolutionary learning to generate neural-network controlled opponents for a complex strategy game called PICOVERSE. The results show that the evolved opponents outperform a manually-scripted opponent. In addition, it is shown that evolved opponents are capable of identifying and exploiting holes in a scripted opponent. We conclude that machine learning is an effective tool to improve the quality of opponent intelligence in computer games.

1 Introduction

The aim of opponents in virtual-world computer games is to provide an entertaining playing experience rather than to defeat the human player at all costs. The quality of the opponent intelligence in games such as computer role-playing games (CRPGs), first-person shooters (FPSs) and strategy games, lies primarily in their ability to exhibit human-like behaviour. This implies that computer-controlled opponents should at least meet the following four requirements: (1) they should not cheat, (2) they should exploit the possibilities offered by the environment, (3) they should learn from mistakes, and (4) they should avoid clearly ineffective behaviour. Opponents in today's computer games, however, have not yet reached this level of behaviour. The appeal of massive online multi-player games stems partly from the fact that computer-controlled opponents often exhibit what has been called "artificial stupidity" [5] rather than artificial intelligence.

In early CRPGs and most of present-day FPSs and strategy games an opponent's behaviour is usually determined by a straightforward script such as "attack the target if it is in range, else move towards the target in a straight line". However, more advanced games contain opponents controlled by large scripts comprising hundreds of complex rules. As any programmer knows, complex programs are likely to contain bugs and unanticipated features. As a consequence, artificially intelligent opponents intended to pose a considerable challenge to a human player often suffer from shortcomings that are easily recognised and exploited. For example, in the CRPG SHADOWS OF AMN (2000; illustrated in figure 1) the dragons, the supposedly toughest opponents in the game,

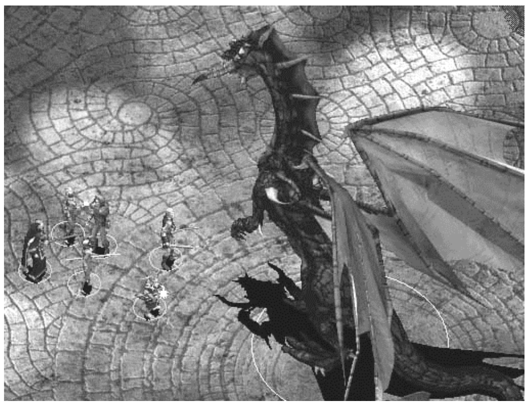


Figure 1: A dragon in SHADOWS OF AMN.

could be easily defeated by taking advantage of holes in the extensive scripts controlling their actions. Evidently, such artificial stupidity spoils the playing experience.

State-of-the-art artificially intelligent opponents lack the ability to learn from experience. Therefore, the research question addressed in this paper reads: How can machine learning techniques be applied to improve the quality of opponent intelligence in virtual world computer games? Section 2 discusses two main ways of applying machine learning to games. Section 3 introduces the strategy game PICOVERSE and outlines the duelling task for which we evolve opponent intelligence. Section 4 describes the environment and techniques we used for our initial experiments. The results are presented in section 5 and discussed in section 6. Section 7 concludes and identifies directions for our future research.

2 Machine Learning for Opponent Intelligence

We distinguish two main ways of applying machine learning to improve the quality of opponent intelligence in virtual world computer games: on-line training and off-line training.

2.1 On-line Training

An example of on-line application of machine learning is encountered in the popular FPS QUAKE. The artificial player in QUAKE III (commonly called a "bot") uses machine learning techniques to adapt to its environment and to select short-term and long-term goals [9]. For QUAKE II, John Laird has developed a bot that predicts player actions and uses these predictions to set ambushes and to avoid traps [3]. Of the four requirements we mentioned in the introduction for opponent strategies that exhibit high entertainment value, these bots address the first two, namely managing to avoid cheating and using their environment effectively. However, they can not learn from mistakes or generate completely new tactics to overcome ineffective behaviour. They mainly adapt to the world they find themselves in, rather than to the tactics of the human player. Still, these bots are a first step towards the creation of human-like opponents by on-line adaptation.

Machine learning techniques are rarely used in commercial computer games. Presumably, the widespread dissatisfaction of game developers with machine learning [10] is caused by the bold aim of creating intelligent opponents using on-line learning. Machine learning requires numerous experiments, generates noisy results, and is computationally intensive. These characteristics make machine learning rather unsuitable for on-line training of opponents in computer games.

2.2 Off-line Training

In the off-line application of machine learning techniques the disadvantages mentioned for on-line learning do not pose an insurmountable problem. However, to our knowledge, developers of commercial games have never used machine learning for off-line learning. In our view the two main applications of off-line learning in games are: (1) to enhance intelligence of opponents by training them against other (scripted) opponents, and (2) to protect opponents against unforeseen player tactics by detecting "holes" in the scripts controlling the opponents. The next three sections describe the off-line training experiments supporting our view on the off-line application of machine learning in games.

3 Duelling Spaceships

In our experiments, we apply off-line training for optimising the performance of opponents in a strategy game called PICOVERSE. This section discusses the game and the learning task to be used in our experiments. Figure 2 shows a screenshot of the game. PICOVERSE is a relatively complex strategy game for the Palm (handheld) computer. Our intentions with the development of this game are twofold: (1) we use it to support and illustrate our views on the design of complex Palm games [7], and (2) in the present context, we use it to investigate the off-line application of machine learning to improve opponent intelligence.

In PICOVERSE the player assumes the role of an owner of a small spaceship in a huge galaxy. Players act by trading goods between planets, going on missions and seeking upgrades for their spaceship. During travel, players encounter other ships and combat may ensue. The ships are equipped with laser guns to fight opponent ships. They are protected from destruction by their hulls. Modelling ship damage, the strength of the hull decreases when hit by laser beams. The duels in PICOVERSE are more strategically oriented than action oriented. While the relative attack power and hull strengths of the spaceships are important factors in deciding the outcome of a fight, even overpowered players have a good chance to escape unharmed if their ship is equipped with fast and flexible drives or specific defence measures. To enhance immersiveness of the game, we permit opponents, who have access to the same equipment as the player, to escape from a duel that they are bound to lose, rather than to continue fighting until being destroyed. This feature makes the opponent intelligence non-trivial, despite the relatively low level of complexity of the game compared to state-of-the-art PC games.

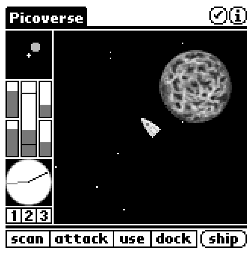


Figure 2: PICOVERSE.

4 Off-line Learning Experiments

In our experiments, the performance of a neural-network controlled spaceship is optimised using off-line training in a simplified version of PICOVERSE. For both the evolved and opponent ships, lasers fire automatically when their enemy is within a certain range and within a 180 degree arc at the front of the ship. Each laser-hit reduces the hull strength of the targeted ship by an amount determined by the quality of the firing laser. If a ship bumps head-on into the other ship, its speed is reduced to zero. The neural controllers are trained using evolutionary algorithms. The fitness is determined by letting the evolved spaceships combat against scripted opponents in a duelling task. Below, we discuss the duelling task (4.1), the neural network controlling the spaceship (4.2) and the evolutionary algorithm (4.3).

4.1 The Duelling Task

Figure 3 is an illustration of the duelling task. We refer to the scripted ship as "the opponent" and to the ship that is controlled by a neural network as "the evolved ship". The scripted behaviour of the opponent is implemented as follows. The opponent starts by increasing its speed to maximum and rotating the ship's nose towards the centre of the evolved ship. While the opponent ship is firing its laser, it attempts to match its speed to the speed of the evolved ship. If the hull strength of the opponent is lower than that of the evolved ship, the opponent ship attempts to flee by turning around and flying away at maximum speed. This simple yet effective script mimics a basic strategy often used in virtual world games.

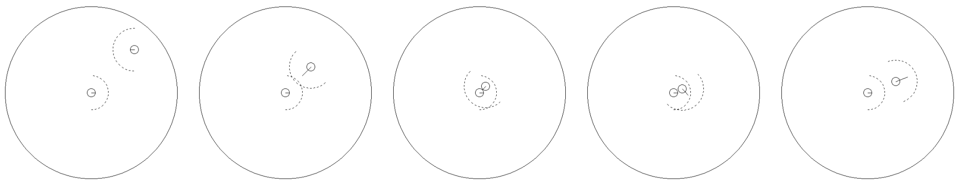


Figure 3: Sequence illustrating the duelling task. The duelling spaceships are represented by the small circles. A ship's direction is indicated by a line inside the circle, its speed by the length of the line extending from the ship's nose. The dotted arc indicates the laser range. The evolved ship is fixed to the centre of the screen and directed to the right. In the sequence the evolved ship is stationary. From left to right, the five pictures show the following events. (1) Starting position. (2) The opponent moves towards the player. (3) The opponent bumps into the other ship. Both ships are firing their lasers. (4) The opponent has decided it should flee and turns around. (5) The opponent flies away and escapes.

4.2 The Neural Controller

The neural network controlling the (to be) evolved ship has ten inputs. Four inputs represent characteristics of the evolved ship: the laser power, the laser range, the hull strength, and the speed. Five inputs represent characteristics of the opponent ship: the location (direction and distance), current hull strength, flying direction, and speed. The tenth input is a random value. The network has two outputs, controlling the acceleration and rotation of the evolved ship. The hidden nodes in the network have a sigmoid activation function. The outputs of the network are scaled to ship-specific maximums.

We studied two types of neural networks, namely feedforward and recurrent networks. The feedforward networks include fully-connected networks (every neuron may be connected to any other neuron, as long as a feedforward flow through the network is guaranteed) and layered networks (neurons are only connected to neurons in the next layer). The recurrent neural networks are layered networks in which recurrent connections are only allowed between nodes within a layer. Recurrent connections function as a memory by propagating activation values from the previous cycle to the target neuron.

4.3 The Evolutionary Algorithm

An evolutionary system, implemented in the ELEGANCE simulation environment [6], was used to determine the neural network connection weights and architecture. All simulations are based on the following settings: a population size of 200, an evolution run of 50 generations, real-valued weight encoding, size-2 tournament selection, elitism, Thierens' method of dealing with competing conventions [8] and size-3 crowding. As genetic operators we used biased weight mutation [4], nodes crossover [4], node existence mutation [6], connectivity mutation [6], and uniform crossover. In addition, we added randomly generated new individuals to prevent premature convergence.

The fitness is defined as the average result of fifty duels between the evolved ship and its opponent. Each duel lasts fifty time steps. Each duel in which the ships started with different characteristics was followed by a duel in which the characteristics were reversed. At time step *t* the fitness is defined as:

$$Fitness_t = \begin{cases} 0 & PH_t \le 0 \\ \left(\dfrac{PH_t}{PH_0}\right) \Big/ \left(\dfrac{PH_t}{PH_0} + \dfrac{OH_t}{OH_0}\right) & PH_t > 0 \end{cases}$$

where PH_t is the hull strength of the evolved ship at time *t* and OH_t is the opponent hull strength at time *t*. The overall fitness for a duel is determined as the average of the fitness values at each time step.

Determining the fitness in this way has the following properties. If the evolved ship and opponent both remain passive the fitness equals 0.5. If the opponent is damaged relatively more than the evolved ship, the fitness is larger than 0.5 and if the reverse is true (or when the evolved ship is destroyed) the fitness is smaller than 0.5. Therefore, the fitness function favours attacking if it leads to victory and favours fleeing otherwise.

5 Results

Table 1 presents the results of the two types of networks tested in the experiments. Evidently, the two-layer feedforward neural networks outperform all other networks in terms of average and maximum fitness values. The network with five nodes in each hidden layer scored only slightly better than the network with ten nodes in each layer.

At first glance the best fitness results achieved are not very impressive. A fitness of 0.5 means that the neural controller results are as effective as the manually-designed algorithm. A fitness of 0.579 (the best result obtained in the experiments) may be taken to indicate that the evolved opponent scores only slightly better than the scripted opponent. Since the scripted opponent employs a fairly straightforward tactic, one would expect the neural controller to be able to learn a more successful tactic. However, a controller that remains passive reaches a fitness of 0.362. Given that a scripted opponent performs at least better than a stationary ship, a fitness of 0.638 is a theoretical upper bound to the maximum the neural controller can reach. From that point of view, a fitness of 0.579 is not bad at all.

Neural network type	*Exps*	*Average*	*Lowest*	*Highest*
Recurrent, 1 layer, 5 hidden nodes	5	0.516	0.459	0.532
Recurrent, 1 layer, 10 hidden nodes	5	0.523	0.497	0.541
Recurrent, 2 layers, 5 nodes per layer	7	0.504	0.482	0.531
Feedforward, 7 hidden nodes	5	0.472	0.382	0.527
Feedforward, 2 layers, 5 nodes per layer	5	**0.541**	0.523	**0.579**
Feedforward, 2 layers, 10 nodes per layer	8	**0.537**	0.498	**0.576**
Feedforward, 3 layers, 5 nodes per layer	7	0.515	0.446	**0.574**

Table 1: Experimental results. From left to right, the columns indicate the type of neural network tested, the number of experiments performed with the neural network, the average fitness, the lowest fitness value and the highest fitness value. The best results are typed in boldface.

From the perspective of playing experience, the fitness rating as calculated in our experiments is not as important as the objective result of a fight. A fight can end in victory, defeat, or a “draw”. For the best controller, we found that 42% of the encounters ended in victory for the evolved ship, 28% in defeat, and 30% in a draw. This means that 72% of the encounters ended in a situation not disadvantageous to the evolved ship, which achieved 50% more victories than the opponent ship. Clearly, the evolved ship performs considerably better than the opponent ship.

6 Discussion

Our results show that machine learning (i.e., off-line training) can be used to create intelligent opponents that outperform scripted ones. Analysing the behaviour of the best-performing spaceship, we observed that it showed appropriate following behaviour when it overpowered the opponent but did not try to flee when it was losing a fight. The probable reason is that for a spaceship to flee, it must turn its back toward the enemy, thereby becoming a target that does not have the ability to fight back (since lasers only fire from the front of the ship). As a result, usually the fleeing ship was destroyed before it could escape. Attempting an escape in these encounters seems therefore of little use. From this observation we learned that if we want to enable escape behaviour, a better balance between the power of the weapons and the versatility of the ships is required.

6.1 Improving the Scripted Opponent

A surprising form of behaviour was observed when the opponent ship starts behind the evolved ship, as illustrated in figure 4. In that case, often the evolved ship attempts to increase the distance between the two ships, up until the moment a draw will occur if it continues increasing the distance. At that point, the evolved ship turns around and either repeats this behaviour a second time or attacks. Figure 5 illustrates the sequence of events. An explanation for the success of the observed behaviour is that if the distance between the two ships is maximal, the evolved ship will have a maximal amount of time to turn around and face the opponent before it gets within the opponent’s laser range. Since facing the opponent is required to counter-attack, this behaviour is beneficial to the evolved ship’s

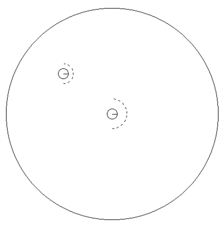
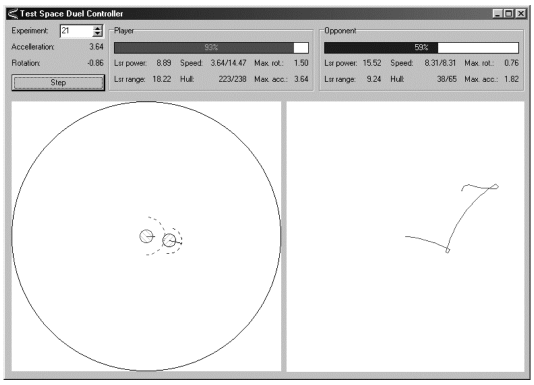


Figure 4: Opponent is behind the evolved ship.

strategy. Therefore, improving the script of the opponent accordingly improves its quality considerably.

By using off-line training, we also detected shortcomings in the scripted opponent. Although we did not specifically design our experiments for this purpose, by observing the behaviour of the two duelling ships, we found a significant hole in the script controlling the opponent. The opponent bases its decision to (attempt to) escape on a comparison between the relative hull strengths. However, it does not take into account the important fact that it is its own turn to act. If on the initial approach the opponent ship gets within the range of the lasers of the evolved ship before it can shoot its own laser, it will be damaged while the evolved ship is still undamaged. Regardless of its own power, this will cause the opponent's initial reaction to be an attempt to escape. We found the evolved ship to exploit this weakness of the opponent. Repairing this hole in the opponent's script will be a major improvement to its behaviour.



```
Experiment: 21
Accelleration: 3.64
Rotation: -0.86

Player
93%
Lsr power: 8.89   Speed: 3.64/14.47   Max. rot.: 1.50
Lsr range: 18.22  Hull:  223/238      Max. acc.: 3.64

Opponent
59%
Lsr power: 15.52  Speed: 8.31/8.31    Max. rot.: 0.76
Lsr range: 9.24   Hull:  38/65        Max. acc.: 1.82
```

Figure 5: The right panel displays a trace of the movements of the evolved ship up to the moment that it fires its first shot. The opponent is overpowered and tries to flee, but the learning ship follows, as shown in the left panel. In this case the opponent is not able to escape.

6.2 Generalisation to Other Games

We have shown how machine learning can be used to improve opponent intelligence in PICOVERSE. Of course, it remains an open question whether our findings generalise to the far more complex commercial PC games. Even the detection of holes in scripted AI, which is obviously much simpler than developing a whole new tactic, may prove too difficult if the number of choices at each turn and the number of turns in an encounter are very large. However, we expect for most games that encounters do not last "too long" (to avoid boredom) and the number of choices is not "too large" (to avoid confusion). Even for most commercial PC games it should therefore usually be possible to detect AI shortcomings by machine learning, allowing the designers to increase the entertainment value of the game by solving these issues before the game is released.

Employing machine learning to design completely new tactics, however, is probably severely limited in its uses. John Laird warns that while neural networks and evolutionary systems may be applied to tune parameters, they are "grossly inadequate when it comes to creating synthetic characters with complex behaviours automatically from scratch" [2]. For a relatively simple game as PICOVERSE machine learning techniques by themselves can be useful in designing strong tactics. The combination of machine learning with more structured techniques, such as a subsumption architecture [1] or a technique inspired by Laird's Soar Quakebot [3], is likely to lead to more reliable good results within a shorter time, and may therefore also be suitable for more complex environments.

7 Conclusions and Future Work

By applying off-line learning in the computer strategy game PICOVERSE we were able to improve opponent intelligence and to detect shortcomings in the scripted opponent. We conclude that the off-line application of machine learning improves the quality of opponent intelligence in virtual world computer games. We expect the automatic detection of holes in commercial computer-game scripts to become feasible by using learning techniques.

Our future research will build upon our results with PICOVERSE. For creating new opponent tactics, we intend to explore other machine learning techniques in combination with, for instance, subsumption architectures. In the long run, we hope to apply our techniques to improve opponent intelligence in commercial computer games.

References

[1] R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139-159, 1991.

[2] John Laird. Bridging the Gap Between Developers & Researchers. *Game Developers Magazine*, August 2000.

[3] John E. Laird. It Knows What You're Going To Do: Adding Anticipation to a Quakebot. *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 385-392, 2001.

[4] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Morgan Kaufman, California, pp. 762-767, 1989.

[5] Jonathan Schaeffer. A Gamut of Games. *AI Magazine*, vol. 22 nr. 3, pp. 29-46, 2001.

[6] P.H.M. Spronck and E.J.H. Kerckhoffs. Using genetic algorithms to design neural reinforcement controllers for simulated plants. *Proceedings of the 11th European Simulation Conference* (eds. A. Kaylan & A. Lehmann), pp. 292-299, 1997.

[7] Pieter Spronck and Jaap van den Herik. Complex Games and Palm Computers. *Entertainment Computing: Technologies and Applications*. Kluwer, 2002 (to be published).

[8] D. Thierens, J. Suykens, J. Vandewalle and B. de Moor. Genetic Weight Optimization of a Feedforward Neural Network Controller. *Artificial Neural Nets and Genetic Algorithms* (eds. R.F. Albrechts, C.R. Reeves and N.C. Steel). Springer-Verlag, New York, pp. 658-663, 1993.

[9] J.P.M. van Waveren and L.J.M. Rothkrantz. Artificial Player for Quake III Arena. *2nd International Conference on Intelligent Games and Simulation GAME-ON 2001* (eds. Quasim Mehdi, Norman Gough and David Al-Dabass). SCS Europe Bvba, pp. 48-55, 2001.

[10] Steven Woodcock. Game AI: The State of the Industry. *Gamasutra*, http://www.gamasutra.com/features/20001101/woodcock_01.htm, 2000.

The program ELEGANCE is available from http://www.cs.unimaas.nl/p.spronck/.
PICOVERSE is targeted for release late in 2002, available from http://www.picoverse.com.