In Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008), pp. 108-113, Stanford, California 2008.

Constructing Complex NPC Behavior via Multi-Objective Neuroevolution

Jacob Schrum and **Risto Miikkulainen**
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA
{schrum2,risto}@cs.utexas.edu

Abstract

It is difficult to discover effective behavior for NPCs automatically. For instance, evolutionary methods can learn sophisticated behaviors based on a single objective, but realistic game playing requires different behaviors at different times. Such complex behavior is difficult to achieve. What is needed are multi-objective methods that reward different behaviors separately, and allow them to be combined to produce multi-modal behavior. While such methods exist, they have not yet been applied to generating multi-modal behavior for NPCs. This paper presents such an application: In a domain with noisy evaluations and contradictory fitness objectives, evolution based on a scalar fitness function is inferior to multi-objective optimization. The multi-objective approach produces agents that excel at the task and develop complex, interesting behaviors.

Introduction

Discovering complex behaviors automatically would be very useful for game development. Game designers could use such methods to train non-player characters (NPCs), or they could train agents against scripted behaviors to discover weaknesses in the scripts. It may even be feasible to train agents against players themselves: The player would have to constantly adapt to learning opponents, which would provide an entertaining and challenging experience.

However, because games are so complex, they often involve multiple objectives. Most learning methods only learn based on a single objective: Reinforcement learning methods learn based on scalar rewards, and evolutionary methods learn based on a scalar fitness function. These methods do not consider the trade-offs between objectives, and when faced with contradictory objectives it becomes particularly difficult to decide how they should be optimized.

In such situations, multi-objective optimization is needed, and in particular, information about trade-offs between objectives. Use of such information is the hallmark of Pareto-based methods (Coello 1999). In this paper, "multi-objective" refers exclusively to Pareto-based methods. Such techniques have been applied to the n-parity problem (Jong, Watson, and Pollack 2001), scheduling (Tamaki, Kita, and Kobayashi 1996), tic-tac-toe (Yau, Teo, and Anthony 2007), and other optimization problems, but they have not been used to discover multi-modal NPC behaviors before. This paper develops such an approach. First, a game is presented that requires multi-objective optimization for NPCs, in order to develop strategies representing trade-offs between objectives. A multi-objective approach to solving it is then described, and shown to be superior to the traditional single-objective approach. The method therefore shows the importance of a multi-objective approach for discovering complex behaviors for NPCs automatically.

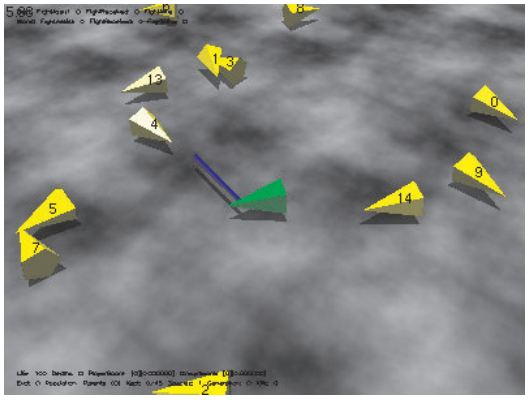

Figure 1: **Untrained monsters in battle domain:** In this game, the player controls the green (dark) agent, and evolved neural networks control the yellow (light) monsters (NPCs). The monsters try to attack the player, and the player tries to hit them with a bat. The challenge is to evolve complex behavior for the monsters that combines several objectives, such as attacking, assisting others in attacking, staying alive, and avoiding getting hit. Initially their behavior is random; successful evolved behavior is shown in figure 3.

Battle Domain

The simulation was developed using BREVE (Klein 2003), a tool supporting all of the most important features of most modern game engines. BREVE is free for download at http://www.spiderland.org.

In the domain (figure 1), a group of evolved NPC "monster" agents faces off against a single "player" agent that uses a "bat" as a weapon. If the player hits a monster with its bat, the monster is knocked back and receives damage. If a monster collides with the player, the player is knocked back and receives damage. The player cannot swing the bat while being knocked back. Once it regains control of itself it is always facing in the direction of the monster that hit it.

The player faces multiple monsters simultaneously, but no more than 15 at a time. A larger population can be evaluated by having several groups of 15. The game consists of repeated trials. At the beginning of a trial, monsters are initialized at random locations surrounding the player. It is then up to the monsters to increase their group score by damaging the player, and to prevent the player from achieving a high score by avoiding getting hit. The monsters and the player move at the same speed, and their environment is an infinite plane. Therefore, at the beginning of each trial, the player is surrounded by monsters to make for an interesting game.

The task is challenging because evaluations are noisy, meaning that a monster's evaluations can each result in different scores. Monsters start at random positions, and because they act in parallel, their actions affect how other monsters are evaluated. A monster can also be unlucky by becoming the focus of the player's attention. To overcome this problem, scores are averaged across multiple evaluations.

Fitness Objectives The domain is also challenging because the monsters must fulfill multiple contradictory objectives. They must avoid being hit by the player's bat, since hits increase the player's score, and eventually kill the monsters. If death is unavoidable, then it should be avoided for as long as possible. If the player can kill monsters quickly, then it can keep shifting its focus to new monsters, and gain more points. Finally, monsters must maximize the damage that they as a group deal to the player. It is not the individual score, but the group score that matters. From the player's perspective, all monsters are the same. Therefore, the monsters must work together to maximize group score. The following three fitness measures are designed to measure success in each of these objectives:

1. *Attack Assist Bonus:* Monsters inflict 10 damage for touching the player. After taking 100 damage the player dies, and the trial continues from the start state. Because the monsters' goal is to maximize group score (as opposed to individual score), a fitness score measuring damage dealt by individuals is inappropriate. Preliminary experiments showed that using this score leads to competitive behaviors that increase an individual's score, but result in poor performance for the population. Therefore, "damage inflicted" was not used as an objective, but a similar score was used instead. Whenever the player receives damage, all monsters within a small radius of the player receive an assistance bonus equal to the damage dealt. The justification is that the actions of nearby monsters influence player behavior and create distractions and opportunities for other monsters. The monster that hit the player receives an additional one bonus point. This scheme rewards monsters that help others inflict damage, but gives an extra reward to those that actually hit the player.
2. *Damage Received:* Monsters start each trial with 50 hit points. A hit from the player's bat deducts 10 hit points; a monster can therefore only be hit 5 times in a trial. After hit points go to 0, the monster dies and is removed.
3. *Time Alive:* A counter is incremented every simulation time step. For each monster, "time alive" is a score from 0 to 900, the total number of iterations per trial.

Though all objectives are important, some take precedence over others. Time alive is least important because it has the least bearing on player and monster scores. The next most important is damage received, because it directly relates to the player's score. Damage received is less important than the assistance bonus because monsters can easily minimize damage received by running away from the player, making for a very boring experience. Minimizing damage received is only important when actually trying to inflict damage.

Such rankings are vital to the single-objective approach described below. In contrast, they are not necessary when applying a multi-objective approach (though the approach used in this paper does make use of them). This freedom from having to provide extra human input is yet another advantage of multi-objective methods.

These objectives are in opposition because the easiest way to reduce damage received and extend time alive is to avoid the player, but gaining a high assist bonus requires monsters to stay near the player. Likewise, a kamikaze attack may result in a higher assist bonus than patiently waiting for the right moment to strike, but it would also result in injury and death. There also exist strategies between these extremes that exemplify the trade-offs in the domain. From a multi-objective standpoint, many different solutions can be considered successful. From a player's perspective, many different solutions/behaviors make the game challenging and fun. Therefore, the battle domain is a useful platform for studying complex behavior: It is simple, yet has contradictory objectives that lead to different types of behaviors.

Evolutionary Method

Given the complex interplay of objectives, it is hard to specify in advance which behaviors exemplify a proper trade-off between them. Any set of rules is likely to represent a single suboptimal solution. Evolutionary methods create populations of solutions to a problem, which allows different solutions to represent different trade-offs between objectives. Specifically, neuroevolution (use of evolutionary methods to create neural networks) has proven to be a good way to learn complex behaviors from single objectives (Nolfi, Parisi, and Elman 1994; Fogel 2002; Stanley, Bryant, and Miikkulainen 2003; 2005), so it makes sense to extend it to work in multi-objective situations.

The neuroevolution method used borrows many ideas from Neuroevolution of Augmenting Topologies (NEAT; Stanley and Miikkulainen 2002). Networks start with a very basic topology. A basic mutation operation that modifies an existing connection weight is used, as well as mutations for modifying network topology by adding neurons and connections. New connections can be recurrent, which allows networks to have memory. On the other hand, unlike in NEAT, crossover, speciation and fitness sharing are not used. Fitness sharing was not implemented because with multiple objectives it is difficult to define. Crossover and speciation were implemented, but preliminary experiments showed that without fitness sharing, these features lead to an overly homogeneous population. While it may be possible to incorporate fitness sharing, crossover and speciation in future work, simply removing these elements makes for a simpler algorithm

that still works well in this task.

To get new offspring, a method similar to the $(\mu + \lambda)$ Evolution Strategy (Bäck, Hoffmeister, and Schwefel 1991) is used: Each parent creates a clone of itself that is then modified via mutations with some small probabilities (each mutation type with a different fixed probability). To progress, elitist selection takes the best half of the combined parent and clone population to be the next parent population, and the rest are thrown away. Which monsters are best depends on whether multiple or single objectives are considered.

Multi-Objective Evolution

To evolve with multiple objectives, a modified version of the well-known multi-objective evolutionary algorithm NSGA-II (Non-dominated Sorting Genetic Algorithm; Deb et al. 2000) is used. The algorithm works by sorting the population into non-dominated Pareto fronts in terms of each individual's fitness scores, in order to select those that are "dominated" by the fewest individuals.

Definition of Domination: $\vec{v} = (v_1, \ldots, v_n)$ dominates $\vec{u} = (u_1, \ldots, u_n)$ iff
1. $\forall i \in \{1, \ldots, n\} : v_i \geq u_i$, and
2. $\exists i \in \{1, \ldots, n\} : v_i > u_i$.

Write $\vec{v} \succ \vec{u}$ to denote that $\vec{v}$ dominates $\vec{u}$. Vector $\vec{v}$ within population $\mathcal{F}$ is said to be non-dominated if there does not exist any $\vec{x} \in \mathcal{F}$ such that $\vec{x} \succ \vec{v}$. The vectors in $\mathcal{F}$ that are non-dominated are called Pareto optimal, and make up the non-dominated Pareto front of $\mathcal{F}$.

NSGA-II selects individuals by first determining which monsters are in the non-dominated Pareto front. It then removes these individuals from consideration momentarily and calculates the non-dominated Pareto front of the remaining members of the population. This process repeats until the whole population is sorted into successive Pareto fronts.

NSGA-II then selects the members from the highest ranked Pareto fronts to be the next parent generation. A cutoff is often reached where the Pareto front under consideration holds more individuals than there are remaining slots in the next parent population. The original NSGA-II selected individuals from this front based on a metric called *crowding distance*. Its purpose was to select a diverse set of solutions from the lowest front so as to maintain as much diversity as possible. However, the version of NSGA-II used in this paper instead uses the priorities between objectives described earlier to aid in selection from the last Pareto front.

This preference information is used in the following manner: When the current front is larger than the number of individuals needed, the scores for the least important fitness objective are simply thrown out, and the members of that front are sorted according to the remaining objectives. This process repeats until the right number have been selected or there is only one objective left, in which case a front can only consist of individuals with identical scores. Any remaining parents needed are picked randomly from that front.

Picking from the last Pareto front in this way steers evolution towards the more important objectives, at the cost of decreased diversity. However, because crossover is not used, the population is rather diverse, and such loss is not a big problem. Although the preference information slightly speeds up evolution, its use is not strictly necessary: The crowding distance metric could be used instead. The preference information was used primarily to provide a fair comparison with the single-objective method described below, and because the ranking of objectives for this particular problem is obvious given the arguments above.

Another issue with NSGA-II is that it depends on reliable fitness information, which is a problem when evaluations are noisy. This problem was overcome by averaging scores over multiple evaluations. There also exists a version of NSGA-II that is specifically designed for noisy domains (Babbar, Lakshmikantha, and Goldberg 2003). Noisy NSGA-II averages across multiple evaluations as done in this paper, but also uses a more lenient selection mechanism to cope with noisy evaluations. However, just averaging across trials was found to be simple and sufficient for the battle domain.

Single-Objective Evolution

The multi-objective approach is compared with a single-objective approach that nonetheless uses all available fitness information. This approach combines all fitness objectives into a scalar fitness value. Elitist selection then picks the best members of the population based on this value.

To combine all fitness objectives, the z-score of each objective score is calculated with respect to the current population of scores for that objective. A score's corresponding z-score is its number of standard deviations away from the mean value of the population. Such standardization allows scores from different distributions to be compared on the same scale. In order to leverage the objective ranking information described above, these z-scores are weighted by coefficients: 1.0 for damage received, 0.5 for time alive, and 2.0 for assist bonus. The sum of these weighted scores is the final fitness score. In contrast to the multi-objective method, this method performs quite poorly on the given task if all objectives are treated equally (i.e. with equal weights).

This fitness function is similar to the one used in NERO (Stanley, Bryant, and Miikkulainen 2005). NERO is a machine-learning game in which the player puts agents through various training tasks for the sake of winning battles against other teams. In NERO, the player can set several fitness preferences via the user interface. NERO uses the same z-score based fitness function, except that the player's fitness preferences define the coefficients for each z-score.

Experimental Approach

To evolve complex behavior in the monster population, monsters faced three increasingly challenging player strategies. These strategies were static, so the word "bot" is used in place of "player" below. Such strategies were necessary to start evolution with an easier challenge so that promising individuals could be identified and evolved further (Gomez and Miikkulainen 1997). Monsters evolved in this manner against the bot can later be evolved further against a human player. The following three bot strategies were used, and represent a logical progression between difficulty levels:

1. *Spinning:* The bot spins in place while swinging its bat. To defeat this strategy, monsters must learn to wait until the bot's back is turned, then rush in and retreat. Quick

retreat is necessary because the bot automatically faces the direction from which it was hit, allowing it to hit any monster that does not quickly retreat.

2. *Alternating:* The bot switches between spinning in place and advancing forward. The bot spins for a short amount of time, and then starts moving forward while swinging in whatever direction it was facing at the end of the spin. Because the bot turns towards any monster that hits it, it will end up approaching any monster that hits it while it is moving forward. Monsters must therefore retreat quicker than before to be successful.
3. *Chasing:* The bot constantly moves forward and always turns to pursue the nearest monster in front of it. If no monsters are in front of it, it keeps turning until it finds a monster to pursue. This strategy is challenging because the bot never turns away from a monster that is threatening it. Monsters in front of the bot have to keep backing away as the bot pursues them. Because monsters and the bot move at the same speed, as long as the bot is chasing some monster, other monsters that are pursuing the bot from behind will generally not be able to catch up with it. Their only hope is to distract the bot somehow to slow it down, or else find an opening to get around its attacks.

To compare the multi-objective and single-objective methods, the simulation was run 30 times using NSGA-II and 30 times using the scalar z-score fitness. Each run consisted of three populations of 15 monsters for a total population of 45 monsters. Each experiment ran for 300 generations. Each generation consisted of five evaluations for the parent population and five evaluations for the clone population over which fitness scores were averaged.

The bot strategies progressed from spinning to alternating, and from there to chasing. The bot switched strategy when the monsters fulfilled three goals:

1. The average amount of damage inflicted by monster groups was consistently over 100, meaning the monsters killed the bot at least once per evaluation.
2. The average amount of damage received per monster was consistently less than 20. Therefore, monsters received no more than two hits on average.
3. The average time alive per monster was consistently over 850, that is, monsters survived nearly the whole time on average.

The word "consistently" is used because the bot should not switch strategies until the monsters have repeatedly demonstrated their ability to fulfill these goals. Good scores on one evaluation can be a fluke, so instead, a recency-weighted average of the score over previous generations must meet the above requirements. Such an average reflects the performance of the population over the past few generations (10 generations in the case of this experiment). It is better than a full average because it does not penalize populations for bad performance many generations ago, yet still requires consistent performance. This way, sudden positive innovations in the evolving population can be rewarded within 10 generations, instead of the many more it would take for a full average to adjust to increased scores. Evolution should not waste time on tasks it has already mastered.

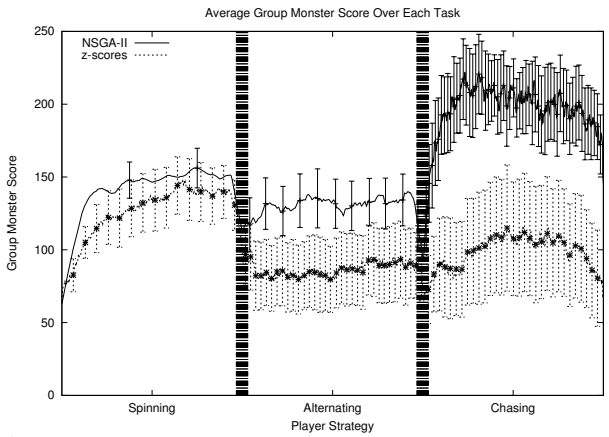

Figure 2: **Incremental evolution; NSGA-II vs z-score method:** The graph plots the average group scores of NSGA-II (solid line) against the z-score method (dotted line) over evolution time. Time to defeat each strategy is stretched to the same length for each bot strategy to provide meaningful averages (values for "missing" generations were interpolated between those of nearby generations). If a task was not reached in a trial, a score of 0 was used for that trial. Error bars represent 95% confidence intervals. The spacing of error bars is denser in tasks where more generations were spent. The figure shows that NSGA-II significantly outperforms the z-score method against the alternating and chasing strategies.

Results

NSGA-II outperformed the z-score method in terms of average group score and in terms of time to defeat the alternating and chasing strategies. There was no significant difference between the two methods against the spinning strategy.

Figure 2 shows averaged monster group scores across the 30 trials. In each trial, each bot strategy was defeated at a different time, so in the figure these intervals are stretched to the same length. As can be seen, both methods perform equally well against the spinning strategy. NSGA-II's scores against the alternating strategy are significantly better than those of the z-score method, but not in all cases. However, NSGA-II is clearly better than the z-score method against the chasing strategy. Differences significant with $p < .05$.

The two methods were also compared in terms of the number of generations taken to beat each bot strategy. This comparison was done using median tests because there were trials that did not beat certain bot strategies within the allotted 300 generations. In failed trials, the number of generations to beat the given strategy is effectively infinity, which makes a comparison of means impossible. NSGA-II always beat the spinning and alternating strategies, but failed to beat the chasing strategy in four trials. The z-score method failed to beat the spinning strategy in three trials, the alternating strategy in six additional trials (nine total), and the chasing strategy in five additional trials (14 total).

There was no significant difference in time to defeat the spinning strategy ($\chi^2(1, n = 60) = 1.06, p = .30$), but NSGA-II was significantly faster than the z-score method against the alternating strategy ($\chi^2(1, n = 60) = 4.29, p = .038$) by a large amount ($\phi = .55$). NSGA-II was also significantly faster against the chasing strategy ($\chi^2(1, n = 60) = 4.27, p = .039$) by a large amount ($\phi = .55$).

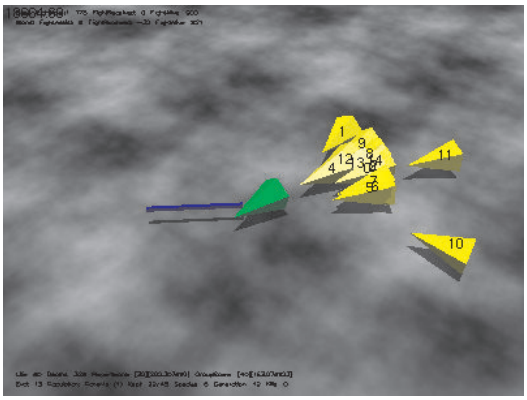

Figure 3: **Trained attack-and-retreat behavior:** The monsters have learned to stay close to the bot, but to avoid its bat. They patiently wait for an opportunity to attack, which comes when the bot turns its back. In this manner, they have learned to be offensive and defensive at different times, demonstrating effective multi-modal behavior. This behavior is effective against the spinning and alternating strategies. Against the chasing strategy, this strategy ultimately evolves into bait and charge strategies. For animations of these strategies, see http://nn.cs.utexas.edu/?multiNPCs.

These results show that multi-objective evolution is able to solve difficult problems faster and more reliably than single-objective evolution.

Observation of evolved behaviors shows that they are not only successful, but complex as well. When evolved against the spinning strategy, monsters learn attack-and-retreat behavior (figure 3). They can sense when they are in front of the bot and when they are within swinging distance. They learn to rush in when the bot turns away. The bot instantly faces any monster that hits it, so after a hit all monsters in range quickly vacate the danger zone before the bot resumes attacking.

Despite not being able to sense the locations of other monsters, the monsters evolve a form of teamwork (role-based cooperation; Yong and Miikkulainen 2007) in which they surround the bot and knock it back and forth so that the bot never gets a chance to swing its bat. Because the bot automatically turns to face whoever hits it, it actually flies back-first towards monsters that are waiting on the other side. These monsters now rush in and hit the bot back towards the monster(s) that originally hit it.

In general, strategies that work against the spinning strategy also work against the alternating strategy, but against the chasing strategy monsters must learn more complex behaviors to succeed. These behaviors were observed primarily in the multi-objective trials. One of the reasons that the z-score method has trouble reaching the chasing strategy is that its fitness function makes exploring new strategies difficult.

To illustrate, consider an NSGA-II population consisting only of monsters that run away from the bot. If a monster in this population rushed the bot, hit it, and then died from a series of bat strikes, it would actually be considered very fit by multi-objective evolution, and would be in the next parent population; despite taking more damage than anyone else, it would have been the only individual that dealt damage to the bot. In contrast, the z-score method would not keep this monster. Because the fitness components are summed together, the positive influence of hurting the bot would be drowned out by the negative influence of dying. In such situations, it is hard for a defensive population to become more aggressive, because most members of the population already have high fitnesses from the damage-received and time-alive components. This is why multi-objective optimization is better than single-objective optimization in complex games. Only by considering multiple objectives separately can all possible avenues to success be rewarded appropriately.

As an illustration of complex multi-modal behaviors evolved in this domain, consider the bot's chasing strategy. Monster populations that were successful against this strategy tended to mix individuals using one of two strategies:

1. *Bait:* The reason that the basic attack-and-retreat strategy does not work against the chasing strategy is that the bot runs away from monsters that pursue it from behind and chases away monsters that are in front of it. As monsters in front of the bot back away, they mostly end up no longer being threatened, because the bot focuses on a single monster. However, by the time these monsters are no longer threatened, the bot has already moved forward enough that they are as far behind as the rest of the crowd following the bot. At this point, the bot is focusing on one monster, whose only guaranteed way to avoid being caught is to move straight away from the bot. However, monsters evolve a clever behavior that ends this stalemate.

   A monster being chased turns by small amounts as it backs up to avoid facing the bot. The bot turns to pursue it, which enables the bot to eventually catch up with the monster, since turning while moving puts less distance between bot and monster than moving straight. However, since the bot also turns to pursue the monster, the monsters chasing the bot eventually catch up. Sometimes the monster being pursued will incur damage as a result of these slight turns, but regardless of whether the bait takes damage or not, any monsters pursuing from behind are eventually able to catch up and attack. One monster takes a risk so that the rest can catch up, which can be seen as an example of evolved altruism (Frank 1994). Once the bot is hit, the monsters are usually able to bounce the bot back and forth for a few hits before it regains enough control to start pursuing a new monster.

   This is a multi-modal behavior that depends on whether a monster is in front of or behind the bot. When acting as the bait, a monster puts itself at risk in order to trap the bot. Multi-objective evolution has found a good balance between objectives in that monsters are willing to risk a little damage in exchange for a higher assist bonus.
2. *Charge:* Although monsters using the baiting strategy avoid damage and do a good job inflicting damage, they waste a lot of simulation time avoiding the bot when they could be hitting it.

   In some simulations a riskier strategy was learned that does not have this problem. It turns out that if the bot moves forward after being hit, the monster that hit it can rush the bot and hit it again in mid-swing. Being hit cancels the bot's swing and knocks it back again. As long as the bot rushes forward it is vulnerable to attack, be-

cause moving forward gives the monster just enough time to reach the bot before the bat reaches the monster.

This charging strategy is very effective against the chasing strategy. A monster can repeatedly bash the bot without it completing a swing. Therefore, against this strategy, the charging behavior deals more damage than the baiting behavior, but charging also leads to monsters being damaged more because of the risk of the initial attack to knock the bot back, and because many monsters are knocking the bot in different directions, which often puts it in a position to strike a monster other than the one that hit it.

Despite resulting in a high group score, charging has some obvious drawbacks that make it less desirable in the general case. Charging only works if the bot is also moving forward. If the bot were to remain still or move backwards after being hit, then charging monsters would not be able to reach the bot before being hit. Therefore, a more successful population is one mixing baiting and charging individuals. Such populations also evolved, and were highly effective. This is another benefit of using a multi-objective evolutionary method: The evolved population can be spread across many points of the trade-off surface between objectives.

The charging strategy is another example of the power of multi-objective optimization. This strategy represents a different trade-off between dealing and receiving damage: The monsters take a chance in order to deal more damage, but also tend to receive more. In this manner, several distinctly different but viable and interesting solutions can be discovered using this approach.

Discussion and Future Work

The results show that multi-objective optimization is a promising way to evolve behaviors for domains with multiple contradictory objectives. In the future, these methods can be extended in several useful ways.

The monsters learned in response to static strategies. When the bot strategy changed, the monster strategy also changed. Evolving NPCs against human players would be better, but the time it takes evolution to adjust to new strategies is too long to hold a player's interest. However, if evolved populations could retain knowledge about previously seen strategies, and learn when to switch between them, then NPCs could be evolved against a wide variety of strategies during development so that they would be very flexible in response to human players later on. An approach similar to milestoning in NERO (D'Silva et al. 2005) might be a possible way to complete such staged learning.

Since human players can adapt very quickly, multi-modal behavior is an effective way to build interesting NPCs for interactive games. Such behavior can best be evaluated when it is clear exactly how many different modes of behavior need to be learned. By designing complex domains with clear task delineations, better learning methods can be developed. If they scale well to many objectives, then game designers can more confidently apply such techniques to complex games with an unknown number of tasks.

Of course, being able to exhibit multiple behaviors is a different problem from knowing when a certain behavior is appropriate. Using the right strategy at the right time is crucial to the success of game AI. Therefore, agents must learn in a variety of contexts, with different objectives associated with each context. If multi-objective methods can be scaled to multiple contexts (each with a different set of objectives), then it should be possible to learn what to do and when to do it simultaneously. NPCs evolved in this manner would be able to learn how to outmaneuver players by learning which countermeasure to apply in a given situation, making for more challenging and interesting game AI than is available today.

Conclusion

Complex multi-modal behavior is difficult to discover even with the most powerful current learning methods. This paper shows that each objective needs to be rewarded separately in order for agents to be successful. It is then possible to optimize each behavior and combine them later into true multi-modal behavior. In the future, multi-objective optimization will lead to more complex agents that will make the game playing experience more entertaining and challenging.

References

Babbar, M.; Lakshmikantha, A.; and Goldberg, D. E. 2003. A modified NSGA-II to solve noisy multiobjective problems. *GECCO* 21–27.

Bäck, T.; Hoffmeister, F.; and Schwefel, H.-P. 1991. A survey of evolution strategies. *ICGA* 2–9.

Coello, C. A. C. 1999. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *KAIS* 129–156.

Deb, K.; Agrawal, S.; Pratab, A.; and Meyarivan, T. 2000. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *PPSN* 849–858.

D'Silva, T.; Janik, R.; Chrien, M.; Stanley, K. O.; and Miikkulainen, R. 2005. Retaining learned behavior during real-time neuroevolution. *AIIDE* 39–44.

Fogel, D. B. 2002. *Blondie24: playing at the edge of AI*. San Francisco, CA, USA: Morgan Kaufmann.

Frank, S. A. 1994. Genetics of mutualism: The evolution of altruism between species. *Journal of Theoretical Biology*.

Gomez, F., and Miikkulainen, R. 1997. Incremental Evolution of Complex General Behavior. *Adapt. Behav.* 5:317–342.

Jong, E. D.; Watson, R.; and Pollack, J. 2001. Reducing bloat and promoting diversity using multi-objective methods. *GECCO*.

Klein, J. 2003. BREVE: a 3D environment for the simulation of decentralized systems and artificial life. *ALIFE* 329–334.

Nolfi, S.; Parisi, D.; and Elman, J. L. 1994. Learning and evolution in neural networks. *Adapt. Behav.* 3(1):5–28.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evol. Comput.* 99–127.

Stanley, K.; Bryant, B.; and Miikkulainen, R. 2003. Evolving adaptive neural networks with and without adaptive synapses. *CEC* 4:2557–2564.

Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Evolving neural network agents in the NERO video game. *CIG*.

Tamaki, H.; Kita, H.; and Kobayashi, S. 1996. Multi-objective optimization by genetic algorithms: a review. *Evol. Comput.*

Yau, Y. J.; Teo, J.; and Anthony, P. 2007. Pareto Evolution and Co-evolution in Cognitive Game AI Synthesis. *EMO* 227–241.

Yong, C. H., and Miikkulainen, R. 2007. Cooperative coevolution of multi-agent systems. Technical Report AI07-338, University of Texas at Austin.