In IEEE Symposium on Computational Intelligence and Games (CIG 2009), pp. 325--332, Milan, Italy, September 2009. (Best Student Paper Award).

Evolving Multi-modal Behavior in NPCs

Jacob Schrum and Risto Miikkulainen, *Senior Member, IEEE*

***Abstract*— Evolution is often successful in generating complex behaviors, but evolving agents that exhibit distinctly different modes of behavior under different circumstances (multi-modal behavior) is both difficult and time consuming. This paper presents a method for encouraging the evolution of multi-modal behavior in agents controlled by artificial neural networks: A network mutation is introduced that adds enough output nodes to the network to create a new output mode. Each output mode completely defines the behavior of the network, but only one mode is chosen at any one time, based on the output values of preference nodes. With such structure, networks are able to produce appropriate outputs for several modes of behavior simultaneously, and arbitrate between them using preference nodes. This mutation makes it easier to discover interesting multi-modal behaviors in the course of neuroevolution.**

I. Introduction

A means for automatically discovering effective behaviors would be useful to game developers. 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. The ultimate goal is to train NPCs against the player, thus requiring the player to constantly adapt to changing opponents. Such an environment would provide a challenging and entertaining experience.

In today's complex games, NPCs have to exhibit multiple different behaviors in order to be successful and believable to human players. Multi-modal behavior means exhibiting distinctly different modes of behavior at different times under different circumstances. Although there are several examples of learning methods that have discovered multi-modal behaviors in games and other complex environments [1][2][3], it is generally difficult to generate such behaviors reliably, particularly as the number of modes grows.

This paper presents a method for encouraging the development of such behaviors using neuroevolution (evolution of artificial neural networks). Neuroevolution has already proven useful in developing interesting behaviors in many challenging control tasks [4][5] and games [6][7].

Furthermore, policy-based reinforcement learning (RL) methods, of which neuroevolution is an example, have been shown to have an advantage over traditional temporal-difference RL in Partially-Observable Markov Decision Process (POMDP) domains [8]. The advantage is particularly significant when networks support recurrent connections, since they provide a mechanism for remembering previous states as part of an internal state. The environment used in this paper, as in many complex games, is a POMDP.

This paper shows how the performance of neuroevolution can be improved in multi-modal domains by allowing the evolved neural networks to have several different sets of output neurons, where each individual set completely defines a potential mode of behavior for the agent. When combined with a method for arbitrating between output modes, this method speeds up neuroevolution, and results in better NPC behaviors in a test game developed for this paper.

First the details of the game are explained, with emphasis on why the game requires multi-modal behavior. Then the details of the evolutionary method used to evolve NPC behaviors are described, along with a discussion of the method that enables multiple output modes. Experimental results are presented next, followed by discussion and conclusion.

II. Fight or Flight Game

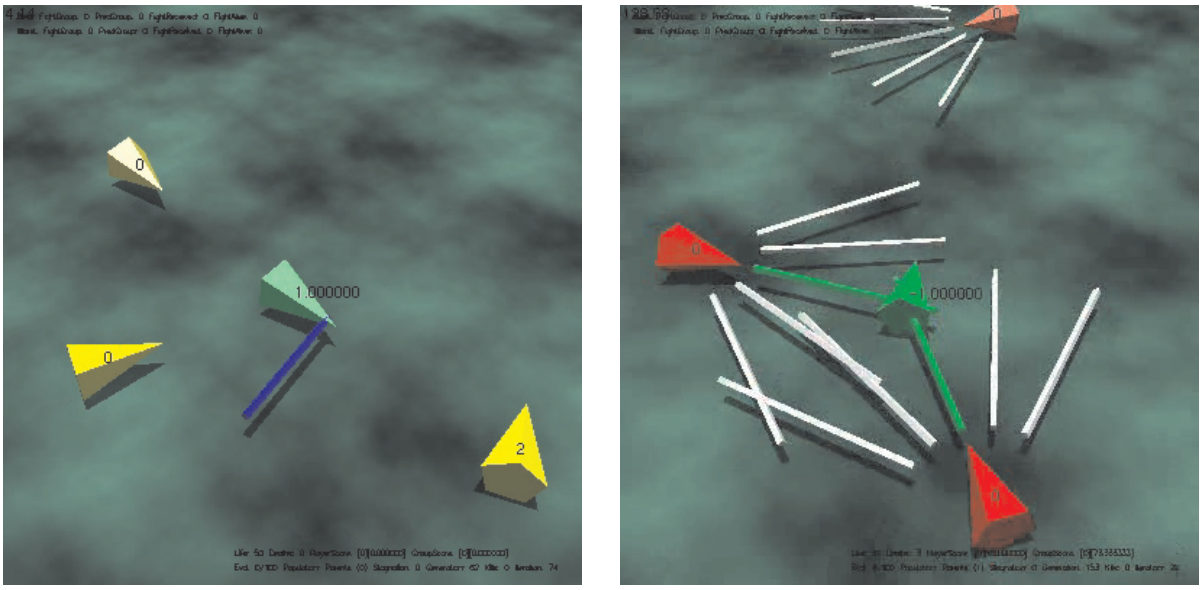
In order to challenge evolution to develop multi-modal behavior, a game called *Fight or Flight* (Fig. 1) was designed in which the player faces two distinct tasks:

- *Fight:* The Fight task is modelled on the Battle Domain presented in [1], but uses a smaller NPC team of only four opponents, thus making the task harder for the NPCs. The player starts in the center of four NPC opponents, which can be attacked with a bat. Striking the NPCs with the bat knocks them back and depletes their health. NPCs can withstand a total of five hits before dying. Similarly, if an NPC comes into contact with the player, the player will be knocked back and will receive damage. The player can also only sustain five hits before dying
- *Flight:* The Flight task shares the same game dynamics as the Fight task, but the player no longer has a bat. Therefore, the four NPCs can damage the player, but are themselves impervious to damage. The player is initially surrounded by four NPCs and must escape without dying. NPCs and the player move at the same speed, so as soon as the player is no longer surrounded by NPCs, the NPCs will not be able to catch it. For this reason, Flight trials end as soon as the player escapes a bounding box defined by the four NPC positions.

NPCs are controlled via artificial neural networks with several input sensors, such as the angles to the player and each teammate, the differences in headings from the player and each of the teammates, an indicator of when the NPC is in front of the player, as well as very brief signals whenever a teammate is injured or the player is damaged. Additionally, each NPC has an array of five sensors spread out in front of it that can detect the player, other NPCs, and the bat. Two network outputs define the agent's behavior: one controls the forward/backward impulse, and a second controls the left/right turning.

In order for the NPCs to be successful, they must exhibit different behaviors in the Fight and Flight tasks. In the

Jacob Schrum and Risto Miikkulainen are with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712 USA (email: {schrum2,risto}@cs.utexas.edu)



a b

Fig. 1. **Fight or Flight Domain:** (a) In the Fight task the NPCs (yellow) surround the player (green). The player has a bat that it swings at the NPCs to deal damage. The NPCs must stay alive and avoid damage, and also inflict damage on the player by colliding with it. (b) In the Flight task the goal of the NPCs (red) is to keep the player surrounded so they can deal damage to it. This image also displays the sensor array used by the NPCs in *both* tasks. In the image, sensor colors change to match the colors of objects with which they come in contact. These two tasks take place in the same environment, and share many sensors, but are different enough that distinct behaviors are required for each task. Therefore, this game is an ideal environment for studying multi-modal behavior.

Fight task they must outmaneuver the player's bat while still finding openings to deal damage. In the Flight task they must keep the player surrounded until they are able to kill it. Multi-modal behavior is also useful within each of these tasks. For example, in the Fight task, the optimal behavior for an NPC when it is in front of the player being attacked vs. behind it should be different. Likewise, in the Flight task, there are different modes of optimal NPC behavior depending on which team member is closest to the player.

The multiple modes of behavior required of the NPCs come about in part because there are multiple objectives in the domain that they must simultaneously optimize. In the Fight task the NPCs must deal damage, avoid damage, and stay alive as long as possible, and in the Flight task the NPCs must also deal damage. To evolve NPCs in this domain, these objectives are measured in the following manner.

1) *Fight - Maximize Damage Dealt:* Every time an NPC contacts the player, the player loses 10 health points. The amount of damage dealt is attributed to the team, regardless of which individual dealt the damage.
2) *Fight - Minimize Damage Received:* Every time the player strikes an NPC with its bat, the NPC takes 10 points of damage, making for a resulting change in health of -10. The fitness attributed to the team is the average change in health across all individuals.
3) *Fight - Maximize Time Alive:* There are 1000 time steps in each Fight trial. For each individual NPC, this objective measures the number of time steps that the NPC is alive. The team score in this objective is the average across team members.
4) *Flight - Maximize Damage Dealt:* The damage dynamics are the same in the Flight case as in the Fight case (10 damage points per collision), but this objective is distinct, i.e. the score is kept separately.

The objectives for the Fight task are similar to those in [1], except the intuitive *Maximize Damage Dealt* objective replaces the comparatively convoluted *Attack Assist Bonus* used in that study. *Attack Assist Bonus* assigned fitness to individuals near the player whenever any NPC dealt damage to it. This measure was used to overcome the team credit assignment problem that results from evolving heterogeneous teams with individual-level selection for a task that requires teamwork. This work uses homogeneous teams with team-level selection since this overcomes the team credit assignment problem and has been shown to be better for tasks requiring teamwork and altruism [9]. Because of team-level selection, all the objectives described above are team-centric.

The *Maximize Damage Dealt* objective for the Flight task is included despite its similarity to the like-named objective in the Fight task because the behavior needed to maximize damage dealt in one task is expected to be different from the behavior needed to accomplish the same goal in a different task. Therefore, the objectives are distinct, and could even potentially be at odds with each other.

Given these multiple objectives, most evolutionary algorithms would require them to be reduced to a single scalar fitness measure, which is typically done using a linear weighted sum. However, there is a class of evolutionary algorithms called Pareto-based Multi-Objective Evolutionary Algorithms (MOEAs) that are specifically designed to deal with multiple objectives without reducing them to a single objective. The main result of [1], which used a domain on

which the Fight task of this paper is based, is that multi-objective evolution performs better than the linear weighted sum approach, so it makes sense to retain multi-objective evolution when adding another task and objective to the mix. The next section describes this evolutionary method.

III. Evolutionary Method

Evolutionary algorithms depend primarily on a means of selecting fit individuals and a means of modifying existing individuals to create new ones. In this paper, a MOEA is used for selection, and Topology & Weight Evolving Artificial Neural Networks (TWEANNs) for modification.

A. 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 [10]) 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: Vector $\vec{v} = (v_1, \ldots, v_n)$ dominates $\vec{u} = (u_1, \ldots, u_n)$ iff

1) $\forall i \in \{1, \ldots, n\} : v_i \geq u_i$, and
2) $\exists i \in \{1, \ldots, n\} : v_i > u_i$.

The expression $\vec{v} \succ \vec{u}$ denotes 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 $\mathcal{F}$.

To get new offspring, a simplified $(\mu + \lambda)$ Evolution Strategy [11] 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.

With NSGA-II, elitist selection is implemented by first determining which NPCs are in the non-dominated Pareto front. NSGA-II then removes these individuals from consideration momentarily and forms a second non-dominated Pareto front based on the remaining members of the population. This process repeats until the whole population is sorted into successive Pareto fronts.

NSGA-II selects the members from the highest ranked Pareto fronts to be the next parent generation. A cutoff is often reached such that the Pareto front under consideration holds more individuals than there are remaining slots in the next parent population. The original NSGA-II selected the remaining individuals from this front based on a metric called *crowding distance*. However, the version of NSGA-II used in this paper, as in [1], is modified to simply re-sort these bottom layers with one less objective each time until the next generation is full.

For example, if 10 more individuals are needed but the current front under consideration contains 20 individuals, one objective will be dropped from consideration and these 20 individuals will be sorted into Pareto fronts using all but the dropped objective. Individuals will be selected and objectives dropped until the needed 10 individuals are selected. If the selection process is reduced to using only one objective, and all individuals have equal scores in this objective, then the last few individuals needed are choosen randomly.

The priorities between objectives are, from most important to least important: Fight-DamageDealt, Flight-DamageDealt, Fight-DamageReceived, and Fight-TimeAlive. The least important objectives are dropped first.

Selection is also improved in this paper by making use of goal information. There is a goal (a numeric level as described later under "Experimental Approach") for each objective that the evolving population must achieve, though it may achieve some at the expense of others. To assure that no subset of objectives takes precedence over the rest, an objective whose goal has already been achieved by the population is dropped from consideration until the population is no longer achieving the goal. The purpose of this mechanism is to prevent the population from being stuck in a situation where certain objectives' scores stay low because performance in the other objectives comes at their expense.

For example, if all members of the population run away from the player during a Fight trial, they will maximize the objectives of avoiding damage and staying alive, but minimize the objective of dealing damage. Despite the fact that any individual in such a population that damages the player will be selected to go to the next generation by NSGA-II, there is a risk of the population stagnating in such a state. Therefore, the objectives for avoiding damage and staying alive are dropped from consideration until such a time that the population is no longer performing well in the corresponding goals for these objectives.

Successive generations will still tend to accomplish goals that previous generations could handle despite dropping objectives for achieved goals, but selection pressure will be focused solely on those objectives whose goals the population is having trouble achieving. Thus stagnation is avoided, and evolution can focus on the most challenging objectives.

B. Neuroevolution

Neuroevolution is the application of an evolutionary algorithm to artificial neural networks. In the Fight or Flight game, these networks are used to control the behavior of NPCs by mapping sensor inputs to NPC actions.

The initial population of networks consists of individuals with no hidden layers, i.e. only input and output nodes. Furthermore, these networks are sparsely connected in a style similar to Feature Selective Neuro-Evolution of Augmenting Topologies (FS-NEAT [12]). Initializing the networks in this way allows them to easily ignore any inputs that are not, or at least not *yet*, useful.

After the cloning stage of NSGA-II, mutations are probabilistically applied to the cloned neural networks (crossover is not used). There are mutations to perturb the weights of existing network connections, add new (potentially recurrent)

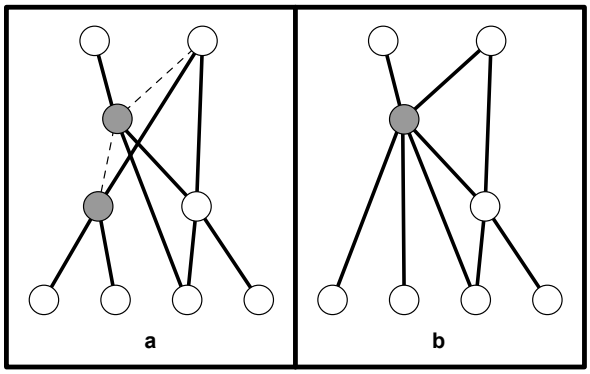


Fig. 2. **Merge Mutation:** (a) the gray nodes are about to be merged. The dashed lines indicate connections that will be deleted by the merge, and the solid lines are connections that will persist. The result of the merge is seen in (b). The mutation reduces network structure without drastically modifying the connectivity of the surrounding nodes.

connections between existing nodes, and splice new nodes along existing connections. These mutation operators are similar to those used in NEAT [13].

An additional mutation operation is introduced in this paper to *reduce* network structure (Fig. 2). This operator performs a merge: A hidden node $\mathcal{A}$ is randomly choosen, and a different random hidden node $\mathcal{B}$ to which $\mathcal{A}$ has a direct connection is also randomly choosen. All connections into $\mathcal{B}$ are redirected to $\mathcal{A}$, unless $\mathcal{A}$ already takes input from the given node, and all outputs from $\mathcal{B}$ become outputs from $\mathcal{A}$, unless such outputs already exist. As the last step both $\mathcal{B}$ and the connection between $\mathcal{A}$ and $\mathcal{B}$ are deleted.

The purpose of the merge mutation is to prevent bloat while damaging fitness as little as possible. There are several neuroevolution algorithms that simplify structure by deleting connections and/or nodes from the network [14][15][16]. This simplifying mutation is unique in that it is less likely to drastically alter network behavior in comparison with a simple deletion, because all nodes to which the merged node connected will receive inputs that should be similar, but slightly different from what they received before.

Preliminary experiments indicate that the merge mutation improves the performance of the algorithm, and leads to more compact solutions.

C. Encouraging Multi-Modal Behavior

The algorithm described in the previous section is modified to encourage multi-modal behavior. However, this proposed methodology can potentially be applied to any neuroevolution method, not just the one presented in this paper.

For a typical neural network to produce multi-modal behavior, it must both detect the need for alternate modes based on the inputs, and use this knowledge to produce optimal (or as good as possible) outputs for each mode. To perform well at a multi-modal task, a network could potentially require the same output nodes to produce very different outputs given inputs that may not be significantly different. For example, in the Fight or Flight game, given that NPCs can sense the player's bat, two nearly identical states, differing only in whether the bat is present or absent, would likely require totally different responses from the NPCs.

In a sense, a domain requiring multi-modal behavior is one that is *fractured* as defined by Kohl and Miikkulainen [17][18]. However, whereas they attack this problem by modifying the types of nodes and connectivity allowed in the hidden layer, in this paper the network is instead provided with several modes of behavior in the output layer. That is, instead of expecting a network to drastically modify its outputs across similar inputs, the network is provided with several sets of output nodes, ideally one corresponding to each mode of behavior. Every output mode consists of a set of output (or policy) nodes capable of completely defining the behavior of the agent. In order to arbitrate between these modes, there is an extra output node for each mode that represents the preference for the given mode. The actions of an agent using such a neural network are defined by first determining which mode has the preference node with the highest value. The policy nodes for the given output mode are then used to define the agent's behavior.

This architecture allows the network to generate outputs for each mode of behavior regardless of which mode is currently the best to execute. Now, in order to act in accordance with the correct mode, only the relative magnitudes of the preference nodes needs to be learned, which should be simpler than having one set of output nodes tuned for every possible mode of behavior. Another benefit of this architecture is that any abstract features about the environment that are discovered within the hidden layer can be shared across multiple output modes, eliminating the need to rediscover useful configurations of hidden nodes more than once.

A potential problem with the algorithm as described thus far is that the number of output modes needs to be known a priori. Even with the knowledge that we have of the Fight or Flight game, it is unclear whether there should be two modes (one per task), four modes (one per objective) or even more. Therefore a method was devised to allow evolution to determine the number of output modes needed.

A mutation is added to the evolutionary method that adds a mode. This mutation adds enough policy nodes to fully define an output behavior, as well as an extra preference node for the new mode. Because mode mutation is potentially a drastic change to the network structure, care needs to be taken to minimize the initial change in network behavior.

To carry out a new mode mutation, the nodes of the last (right-most) output mode $\mathcal{M}$ are first identified, and an equal number of new nodes for the new output mode $\mathcal{N}$ are then created to the right of $\mathcal{M}$. For each node in $\mathcal{M}$, a feed-forward connection with a weight of 1 is added to the corresponding new node in $\mathcal{N}$ (feed-forward connections are allowed within the same layer provided they go from left to right). The outputs of the new mode will thus be similar to those of a pre-existing mode (Fig. 3).

Using the new node mutation, it is possible to learn the optimal number of output modes incrementally while learning how to solve the task.

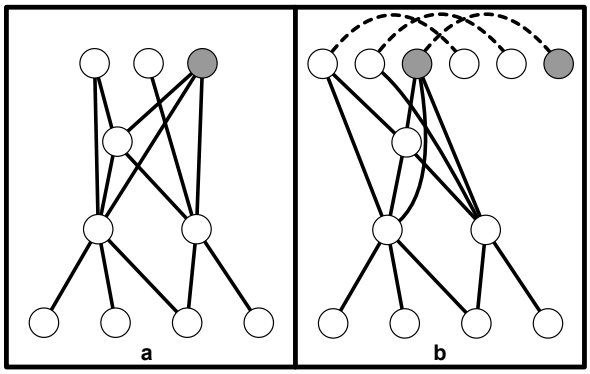


Fig. 3. **New Mode Mutation:** The starting network (a) has two policy nodes and one preference node (in gray). For each of these nodes a new node is created to the right that receives feed-forward input from it along a connection with weight 1 (dashed lines). The resulting network is (b). Future mutations can now create connections to the nodes of the new output mode, and refine it into a distinct and useful behavior.

IV. Experimental Approach

The NPCs are evolved in homogeneous teams using team-level selection. A homogeneous team is created by taking a single neural network to be evaluated, copying it four times, and assigning one copy to each of the four NPCs. Homogeneous teams tend to be better at evolving teamwork because every individual implicitly knows what to expect of its teammates [9]. However, such implicit awareness of one's teammates has also been developed in heterogeneous populations, often by representing the entire team by a single genome [19] or by having each teammember come from a separate subpopulation [20].

With team-level selection the fitness scores are calculated for the team as a whole, and assigned to the genome. Such selection makes sense in environments requiring teamwork, particularly if altruism is necessary [9], since team-level selection will tolerate the suffering of individuals when it serves the greater good of the team.

Neuroevolution combined with the modified version of NSGA-II is used to evolve NPCs using the four NPC objectives listed in the domain description above. A single parent population contains 50 neural networks. Fitness evaluation is noisy, so each neural network is evaluated five times in the domain. A network's final fitness scores are averages of the results from the five evaluations. Participating in five trials means that each network is involved in five Fight trials and five Flight trials, though the agents are not explicitly aware of which type of trial they are facing. Five trials was found to be a reasonable trade-off in terms of reducing noisiness and reducing overall evaluation time.

In order to evolve the NPCs, opponents are necessary. Because human interest cannot be maintained for the many game trials in takes evolution to learn interesting behaviors, the NPCs are evolved against a computer-controlled player. To make this distinction clear, this computer-controlled entity will be referred to as a *bot*.

The bot uses a separate strategy for each of the two tasks of the Fight or Flight game:

- *Fight:* The strategy is the same as the Chasing strategy used in [1]: The bot moves forward and turns towards the nearest NPC in front of it. The bot swings its bat constantly while moving.
- *Flight:* The bot's goal is to escape the NPCs, so it moves backwards away from the nearest NPC that is in front of it (that it can *see*). An advantage of moving backward is that the bot will always end up facing its attacker every time it is knocked back by contact with an NPC. Facing the source of the attack makes it easy for the bot to avoid its most recent attacker by continuing to move backwards after being involuntarily knocked backwards. This bot behavior makes the task harder for the NPCs: they should only hit the bot if doing so will knock it into the midst of other NPCs.

These are challenging strategies for the NPCs. In fact, an initially random population does not have much of a chance to evolve interesting behavior against a bot using these strategies and moving at full speed. Therefore, the bot is initially handicapped by having its speed reduced, and incremental evolution [21] is used to gradually increase the speed whenever the NPC population demonstrates that it is able to handle the bot at the current speed.

Progression is based on goals; there is one goal for each objective. These are the same goals that the modified NSGA-II uses to drop objectives whenever their goals are achieved. A goal is simply a numeric value for an objective that should be attained by an average performing member of the population. Throughout the course of evolution, the average value of each objectve across the successive parent populations is tracked. Additionally, for each objective a recency-weighted average of these average values is maintained.

The purpose of the recency-weighted average is to track the average performance of the population over the most recent evaluations. When the value of a recency-weighted average has surpassed its objective's goal value, the goal is considered achieved. Once all goals are achieved, the difficulty of the task is increased by increasing the speed of the bot. The goals for each objective are:

1) *Fight - Maximize Damage Dealt: 50:* The bot has 50 health points, so this goal requires the NPCs to kill the bot at least once per trial. The bot respawns after death, giving the NPCs a chance to inflict more damage.
2) *Fight - Minimize Damage Received: -20:* Bat strikes deal 10 damage points each, so each NPC should take no more than 2 hits on average. However, because this value is averaged across team members, it is possible to achieve this goal even if one team member dies (50 damage), since the average across the four team members could still be above -20.
3) *Fight - Maximize Time Alive: 800:* On average, team members must survive throughout 80% of the trial. This number is an average across team members as well, so it is still possible to achieve this goal if some NPCs die in fewer than 800 iterations.
4) *Flight - Maximize Damage Dealt: 100:* This goal

requires the NPCs to kill the bot twice per Flight trial. This amount is greater than the amount for the similar goal for the Fight task because without the bat, the NPCs should be able to deal a greater amount of damage in the same time.

Using averages is better than using the generation-to-generation values themselves because these values fluctuate significantly. The recency-weighted average is better than a regular average because it does not punish the population for bad performance in distant earlier generations. The particular α weight used for the recency-weighted average was 0.15, meaning that every update to the average moves it $0.15D$ towards the most recent data point, where D is the difference between the data point and the previous average.

At the start of the simulation, and whenever the population achieves all goals, thus causing the speed of the bot to change, the recency-weighted averages are reset to the minimum values of the corresponding objectives, which are zero for all objectives except Fight-DamageReceived, which has a minimum of -50.

Two different experimental conditions with labels 1Mode and ModeMutation are compared. The 1Mode experiments use neural networks with a single output mode (two nodes). No preference node is needed because there is only one mode. The ModeMutation experiments start with networks that have only one output mode, but that can gain more via the mode mutation. For the ModeMutation trials, a single output mode requires three output nodes (two policy nodes and one preference node).

Each condition is evaluated in 10 separate trials for 300 generations or until all goals are achieved when the bot speed is 100%, whichever comes first. The bot speed starts at 0% which allows for no movement other than turning. When all goals are achieved at this speed, the speed is increased to 40%, then 80%, and finally 100%. Preliminary experiments were done with other speed sequences, and this sequence was found to allow for a reasonably good trade-off between differences and similarities of consecutive speeds.

V. Results

The Fight or Flight game proved challenging for both experimental conditions, but the ModeMutation method was twice as successful as the 1Mode method in that four of the ten trials defeated the bot at 100% speed whereas only two of the 1Mode trials defeated the bot at 100% speed. The number of generations spent by each trial against the bot at each speed is shown in Fig. 4.

However, to fully understand the performance of these methods, the behaviors of the resulting NPCs must be observed. The main theme is that individuals in ModeMutation populations tend to perform well in all objectives, whereas individuals in 1Mode trials tend to focus on certain objectives at the expense of the others (Fig. 5).

More specifically, in order to perform well, evolution should discover multi-modal behavior. For instance, in the Fight task, an established [1] good behavior is a baiting tactic,

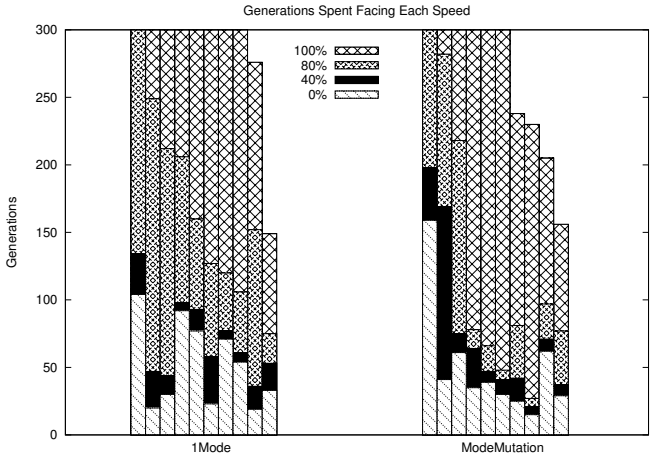


Fig. 4. **Generations Spent Facing the Bot at Each Speed:** For the ten 1Mode trials and ten ModeMutation trials, the generations spent facing the bot at 0%, 40%, 80% and 100% speed are shown in a stacked bar chart. Four ModeMutation trials and two 1Mode trials finished before reaching the 300 generation cutoff.

by which one NPC is chased by the bot and the others follow behind. At 100% speed, it would be impossible for the bot to catch the bait, and the other NPCs to catch the bot, because no one agent would be able to overtake any of the others. However, if the baiting NPC learns to turn slightly to the side while running away from the bot, the bot will turn to chase it, which allows the chasing NPCs to catch up and attack the bot. The bait often suffers some damage as a result of being chased, but the behavior ultimately benefits the team enough to offset this loss.

A good behavior in the Flight task is a corralling tactic in which the NPCs surround the bot and repeatedly knock it towards the center. NPCs must be careful to spread out so that none of them knocks the bot outside of the corral.

Such successful behavior emerged reliably in the four ModeMutation trials that achieved all goals within the allotted 300 generations. One example is illustrated in Fig. 5a; movies can be seen at http://nn.cs.utexas.edu/?multimodal09. In the Fight task, there are at least two modes of behavior being demonstrated: baiting and chasing. The NPCs are using one set of output modes to perform baiting, and then a different output mode to perform chasing. One evolved output mode corresponds to chasing, and two modes are associated with the baiting behavior.

In the right side of Fig. 5a, the exact same team of agents is portrayed in a Flight trial. They now demonstrate the corralling behavior. To prevent escape, the NPCs only knock the bot towards the center. By moving to block the bot, they sometimes trick it into moving back into being surrounded. Interestingly, this behavior depends on the same output mode that gives rise to the chasing behavior in Fight trials. Evolution adapted the common aspects of one output mode into two different tasks, and evolved extra modes to handle the additional behaviors needed by the Fight task.

In contrast, individuals from 1Mode trials, even though deemed successful, tend to be narrowly focused. None of the individuals developed through the 1Mode method excel

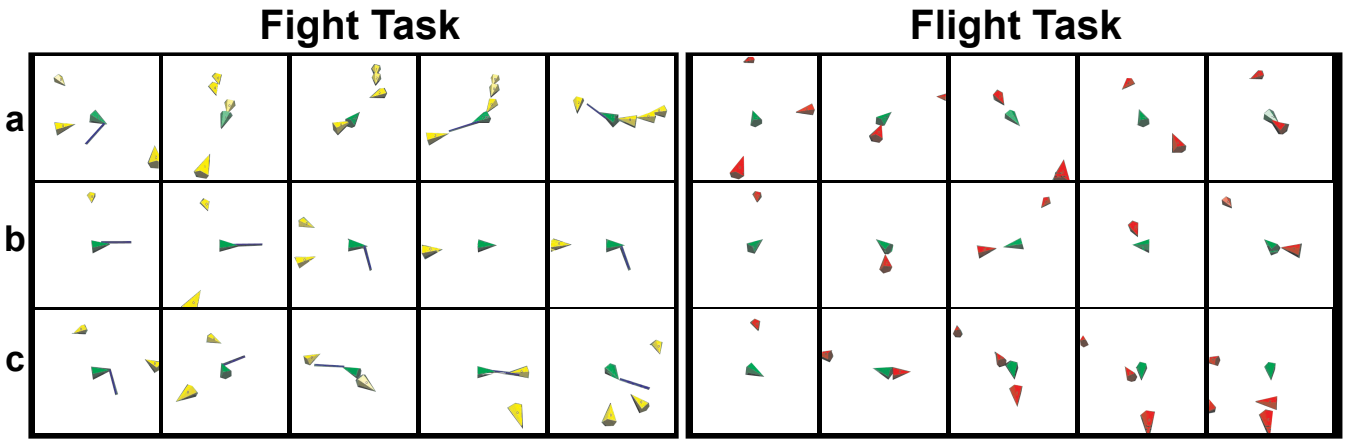


Fig. 5. **Evolved Behaviors:** Each strip shows five scenes from a Fight trial of a given network's evaluation (left) followed by five scenes from a Flight trial (right) faced by the same network. These examples can be seen as animations at http://nn.cs.utexas.edu/?multimodal09. The texture of the floor has been removed to increase visibility in the static display above. (a) A ModeMutation trial in which the NPCs do well in both the Fight task and the Flight task. In the Fight task the NPCs exhibit the baiting strategy, and in the Flight task the NPCs knock the bot to the center whenever it is about to escape. (b) A 1Mode trial in which the NPCs deal no damage to the player in the Fight task. Three NPCs (two of which are visible in the frames) chase the bot while the other runs away from it off screen, and no damage occurs. However, the NPCs do perform well in the Flight trial by knocking the bot back into the center whenever it is about to escape. (c) A 1Mode trial in which the NPCs exhibit a strategy similar to the baiting tactic, but less effective in that they wind up in front of the bot, where they are repeatedly struck with the bat (last two frames of Fight task). These NPCs achieve the goal of killing the bot once, but are so inefficient that all of them die as well. The NPCs also exhibit poor behavior in the Flight task by letting the bot escape after three hits. In the successful trials, ModeMutation discovers networks that perform well across all objectives in both tasks, in contrast to the 1Mode trials that result in networks that only do well in particular objectives by performing poorly in others.

at all objectives. Instead, there is always an imbalance.

For example, the team of 1Mode NPCs portrayed in Fig. 5b ignore the objective to damage the bot in order to do well in the other objectives. In the Fight trial, the NPCs develop a simple behavior that avoids all contact with the bot. They do no damage to it, though they also take no damage and manage to live through the whole trial. However, the NPCs do well in the Flight trial that follows and exhibit corralling behavior. Doing damage in the Fight trial is sacrificed for the sake of the other objectives.

In another 1Mode team (Fig. 5c), NPCs use a strategy like the baiting behavior described above, except that it results in the deaths of all the NPCs. They meet the goal of dealing 50 points of damage to the bot, but do poorly at avoiding damage and staying alive. In the Flight trial the NPCs let the bot escape after just three hits.

Thus, the individuals in the 1Mode trials tend to have extreme scores, but rarely have high scores across all objectives. Because the goal-based progress mechanism used in this paper is based on the average scores of individual objectives, it does not care whether there are any agents doing well in all objectives. Therefore the same goals can be met both in the case where all agents do reasonably well in all objectives (as with the ModeMutation results) and in the case where different agents do extremely well in certain objectives but very poorly in others (as with the 1Mode trials). In fact, the final maximum, minimum and average scores for the successful 1Mode trials are similar to scores for the successful ModeMutation trials. The difference is at the level of individuals rather than the level of the population.

Since we do not know the shape of the true Pareto front for the Fight or Flight game, we cannot know which of these outcomes most accurrately matches the true front. Given that MOEAs are designed to produce a population that is spread across the trade-off surface between objectives, the 1Mode method may be better at this than ModeMutation. However, if the ultimate purpose is to design agents that exhibit multi-modal behavior, then ModeMutation is clearly preferable.

VI. Discussion and Future Work

The experiments in this paper have demonstrated that the ModeMutation method is better suited than the 1Mode method to evolving agents that exhibit multi-modal behavior. One problem with multi-objective evolution is that it considers a population with individuals occupying extreme edges of the trade-off surface to be beneficial. Individuals that maximize some objectives while doing very poorly in others do belong in the Pareto front, but are usually not very good as game NPCs. A way needs to be developed to take advantage of the benefits of multi-objective evolution while focusing on solutions that do well across all objectives. ModeMutation biases evolutionary search towards these solutions.

However, many MOEAs besides NSGA-II exist, and some of them may be able to address this problem as well. For example, instead of performing selection based on which Pareto front a solution occupies, it could be performed based on how many individuals a candidate solution Pareto-dominates, as is done in the Strength Pareto Evolutionary Algorithm [22]. This selection method would bias selection towards solutions that did well in many objectives, and away from solutions near the extreme edges of the Pareto front.

One possible way to improve the current approach would be to change how incremental evolution progresses. Progress goals that are based on average values for each objective

allowed the undesirable solutions of the 1Mode method to be considered successful. It might be better to increase difficulty only when individuals within the population simultaneously meet several objective goals. Such goals would likely prevent bad solutions like those of the 1Mode method from progressing to harder challenges.

However, it might be possible to take advantage of diverse populations with specialized individuals by treating the population as an ensemble. For any given situation, at least one member of such a diverse population would likely exhibit suitable behavior. By learning to arbitrate between the outputs of all members of the population (as in [5]) a single agent could be designed that would exhibit appropriate behaviors in different situations.

There is also room for improvement of ModeMutation itself. For example, the ModeMutation network depicted in Fig. 5a had seven output modes, but only three of these were ever selected to define agent behavior. Some successful ModeMutation networks had as many as ten total modes. In some cases these seemingly superfluous modes *do* serve a purpose, because they have recurrent outputs and/or feed forward outputs to other modes. It is also not certain whether these unused modes are simply evolutionary baggage (never been used) or if they are vestigial (once useful, but no longer so). There are likely ways to apply ModeMutation more intelligently to prevent this kind of bloat, and assure that new modes only arise if they are actually useful.

In the meantime, the ModeMutation used in this paper is a simple and effective method to learn multi-modal behavior. In the future, it should be possible to scale it up to harder domains and more objectives. Furthermore, the method could be used to facilitate transfer learning. An agent could evolve new output modes on top of old modes in order to accomplish new tasks. Such applications constitute an interesting direction for future work.

VII. Conclusion

A mutation operator was proposed for neuroevolution algorithms that aims at producing multi-modal behavior. This operator adds new modes by adding to the output layer of the network.

The approach was applied to a game with two separate tasks and four competing objectives, resulting in NPCs that excel in both tasks. The method is thus a promising starting point for discovering multi-modal behavior and for developing complex agents in challenging tasks in general.

Acknowledgments

This work was supported in part by the National Science Foundation under grant no. EIA-0303609.

References

[1] J. Schrum and R. Miikkulainen, "Constructing complex NPC behavior via multi-objective neuroevolution," in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008, pp. 108–113. [Online]. Available: http://nn.cs.utexas.edu/keyword?schrum:aiide08

[2] B. D. Bryant and R. Miikkulainen, "Neuroevolution for adaptive teams," in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 3. Piscataway, NJ: IEEE, 2003, pp. 2194–2201. [Online]. Available: http://nn.cs.utexas.edu/keyword?bryant:cec03

[3] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for RoboCup-soccer keepaway," *Adaptive Behavior*, vol. 13, pp. 165–188, 2005.

[4] F. Gomez and R. Miikkulainen, "Active guidance for a finless rocket using neuroevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*. San Francisco: Morgan Kaufmann, 2003, pp. 2084–2095. [Online]. Available: http://nn.cs.utexas.edu/keyword?gomez:gecco03

[5] D. Pardoe, M. Ryoo, and R. Miikkulainen, "Evolving neural network ensembles for control problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005. [Online]. Available: http://nn.cs.utexas.edu/keyword?pardoe:gecco05

[6] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[7] K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen, "Real-time evolution of neural networks in the NERO video game," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006. [Online]. Available: http://nn.cs.utexas.edu/keyword?stanley:aaai06

[8] S. Kalyanakrishnan and P. Stone, "An empirical analysis of value function-based and policy search reinforcement learning," in *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

[9] M. Waibel, L. Keller, and D. Floreano, "Genetic Team Composition and Level of Selection in the Evolution of Multi-Agent Systems," *IEEE Transactions on Evolutionary Computation*, 2009, to appear.

[10] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," *PPSN VI*, pp. 849–858, 2000.

[11] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 2–9.

[12] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl, "Automatic feature selection in neuroevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005. [Online]. Available: http://nn.cs.utexas.edu/keyword?whiteson:gecco05

[13] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/keyword?stanley:ec02

[14] D. James and P. Tucker, "A comparative analysis of simplification and complexification in the evolution of neural network topologies," in *GECCO*, 2004.

[15] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 54–65, 1994.

[16] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, May 1997.

[17] N. Kohl and R. Miikkulainen, "Evolving neural networks for fractured domains," in *Proceedings of the Genetic and Evolutionary Computation Conference*, July 2008, pp. 1405–1412. [Online]. Available: http://nn.cs.utexas.edu/keyword?kohl:gecco08

[18] N. Kohl and R. Miikkulainen, "Evolving neural networks for strategic decision-making problems," *Neural Networks, Special issue on Goal-Directed Neural Systems*, 2009.

[19] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning," in *GECCO*, 2008.

[20] C. H. Yong and R. Miikkulainen, "Cooperative coevolution of multi-agent systems," Department of Computer Sciences, The University of Texas at Austin, Tech. Rep. AI07-338, 2007. [Online]. Available: http://nn.cs.utexas.edu/keyword?yong:utcstr07

[21] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, vol. 5, pp. 317–342, 1997. [Online]. Available: http://nn.cs.utexas.edu/keyword?gomez:ab97

[22] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach." *IEEE Transactions on Evolutionary Computation*, 1999.