8 Interactive Neuroevolution

The previous two chapters discussed how the behavior of agents that operate embedded in an environment can be discovered through neuroevolution. Starting from reactive control and expanding all the way to sequential decision making strategies, effective solutions can be discovered that may be surprising to human designers. Moreover, discovery can be embedded in a collective environment, where opponents and cooperators are evolving as well, thereby providing new and creative challenges. In some cases, however, it may be useful for human designers to drive this discovery process more explicitly. They may have knowledge that's difficult to capture in a formal objective function. For instance, the desired behavior may be complex and multifaceted, or depend on believability or esthetic values. In such cases, neuroevolution can be made interactive. The construction of new individuals is still done through evolutionary operators, but the selection is at least partially due to human judgement. This chapter reviews how interactive neuroevolution can be set up effectively, and demonstrates it in several examples in various game domains.

8.1 The NERO Machine Learning Game

Setting up neuroevolution experiments sometimes feels like a game. You have a goal in mind, i.e. an idea what you want the evolved agents to do. You have to think about how to express that behavior in terms of objective function, which in turn depends on behavioral descriptors that can be readily measured. You may need to come up with a shaping strategy, starting with simpler behaviors and gradually making the objective function more demanding. You may need to try out many different such setups before finding some that achieve effective behavior. There may be several such solutions, and some of them may even surprise you. Finding such solutions, and perhaps better than those seen before, is what makes this game appealing.

NERO (Stanley, Bryant, and Miikkulainen 2005) is an actual game build on this very idea. It can be seen as a pioneering effort to establish a new game genre, machine learning games. Unlike in other genres, such as first-person shooter games or sims, the human player is not controlling game agents directly. Instead, the player takes the role of a teacher/coach/drill sargent, designing a curriculum of learning challenges for actual agents in the game. Those agents solve the challenges using machine learning. After learning, the agents engage in a head-to-head competition with other similarly trained agents, in order to determine how good the training was.

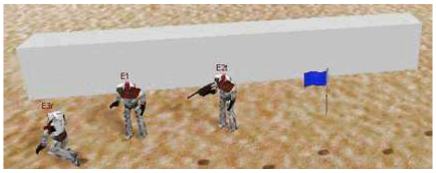
More specifically, in NERO the game agents are battle robots controlled by neural networks evolved with NEAT (Figure 8.1*c*,*d*). The entire population of them is placed in the environment at once. The environment is usually an enclosed area with walls, buildings, trees, and other objects, allowing the agents to move around, hide, and take cover. Simple algorithmically controlled enemy agents can be placed in it, including static enemies (and flags) that act as targets, static enemies that fire at the agents, and mobile enemies that fire and approach the agents. As their input, they observe the number and distance to enemy agents as well as teammates in sectors around them, distance to walls and other static objects in several directions, whether their weapon is on target, and the direction from which the fire from the nearest enemy is coming. As their output, they can move forward and back, turn left and right, and fire their weapon.

In such an environment, NEAT can evolve networks that exhibit interesting behaviors. The agents can charge the enemy, approach from different directions, disperse in order to be less likely to hit, converge to increase firepower, take temporary cover behind walls, hide in order to survive until the end of the game, and many others. The interesting question is: what kind of behaviors are useful in a battle against an actual enemy? Further, how can we encourage evolution to discover such behaviors, while still encouraging open innovation as well? This is precisely the question interactive neuroevolution aims to address.

In NERO, the human player has a number of tools at their disposal (Figure 8.1*a*,*b*). They can place various objects in the field, such as walls, static and mobile enemies, and flags. They can control a number of sliders that correspond to coefficients in the objective function, such as approach/avoid the enemy, hit a target, avoid getting hit, follow teammates, disperse, etc. Both objects and sliders can be changed dynamically as the training progresses, therefore making it possible to design a curriculum. For instance, it is may be useful to reward the agents for approaching the enemy first, then do it while avoiding fire, then while avoiding fire from moving enemies, then while utilizing walls as cover, etc. (Figure 8.2). Such curricular evolution, or shaping, can result in more complex and effective behaviors than could be achieved with a single static objective function without human guidance.

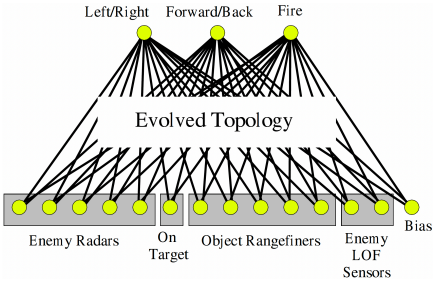
One interesting extension needs to be made to the NEAT method, however. Note that the entire population is evaluated in the environment at the same time. This approach makes the evolution efficient, since the evaluations are done in parallel. The population is also always visible to the human player, making it easier to understand how well the evolution is progressing. However, if the entire population is replaced at the same time, as is usual in generational evolution, the game appears discontinuous and difficult to follow. Instead, evolution needs to progress continously one agent at a time.

In this real-time extension to of NEAT, called rtNEAT, among all the agents that have been evaluated sufficiently long, the worst agent is removed from the population. The species are recalculated and an offspring generated as usual in NEAT. This offspring is then placed in the environment to be evluated. This replacement takes place at regular intervals, and because it involves only one individual at a time, is largely invisible to the human player. In this manner, evolution progresses continuously while the population is being evaluated. Although it was designed for the visual effect in NERO, the same approach can be useful in other domains where continuous adaptation is needed.

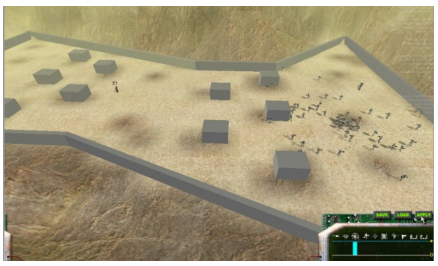


(*c*) Possible objects

(*b*) Sliders defining fitness



(*c*) A network controlling one agent



(*d*) A population being evaluated

Figure 8.1: **Setting up a NERO experiment.** The NERO game allows specifying increasingly challenging environments so that complex behavior can be evolved. (*a*) The human player can place various objects in the environment to create challenges, including walls, flags, static enemies, and moving enemies. (*b*) The human player controls the fitness by adjusting sliders with continuous positive or negative values along various dimensions such as approach enemy, approach flag, hit target, avoid getting hit, and stay together with teammates. (*c*) Each agent in the game is controlled by a neural network evolved through NEAT. As its input, it senses the environment around it, including enemies, teammates, walls and other objects; it also senses whether its weapon is on target, and the direction from which the nearest fire is coming. As its output, it issues actions to move forward and back, turn left and right, and fire. (*d*) During evolution, the entire population of agents is evaluated together in an enclosed environment that may contain multiple objects. In this case the agents spawn on the right and are rewarded for approaching the flag on the left. At regular intervals, the worst agent is replaced by offspring in a continous replacement process. In this manner, the human player can create a curriculum of increasingly challenging tasks that prepares the team well for battle against other teams. For animations of various training scenarios, see https://neuroevolutionbook.com/neuroevolution-demos. (figures from Stanley, Bryant, and Miikkulainen 2005)

After the curricular evolution is complete, the teams are evaluated in a battle mode of NERO. Two teams are placed in the same environment which may be the same used in training, or something completely different. At this stage (in NERO 1.0) the agents operate independently of the human player, applying what they were trained to do in competition with another team. If an agent is hit a sufficient number of times, it is removed from the environment. The game ends when one team is annihilated, or the clock runs out, in which case the team with most agents still on the field wins. Note that the battle domain is obviously a violent game, similar to many video games in the first-person shooter genre. The principles are more general, however, and apply to other less violent settings as well, such as the gardening game of Petalz (Risi et al. 2016). A robotic battle domain, however, provides

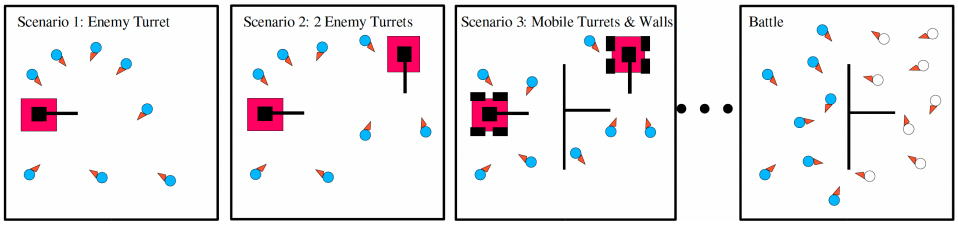


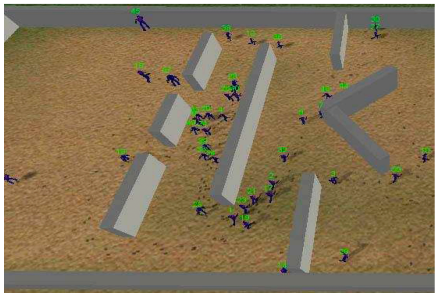
Figure 8.2: **Training NERO teams through interactive neuroevolution.** The player first specifies a simple task such as approaching a static enemy that fires (a "turret"), so the agents learn to approach it from different sides. In the next scenario, they learn to approach one turret while minding fire from another. Next, the turrets move and turn, and the agents need to take cover behind walls. Through multiple such increasingly challenging senarios, the agents learn effective battle behaviors. The team is then placed into a battle against another team, evaluating how well the human player was able to train them. NERO thus aims at creating intelligent behavior strategies through interactive neuroevolution. (Figure from Stanley, Bryant, and Miikkulainen 2005)

clear and compelling measures and visualizations of performance, which was useful for a pioneering example of machine learning games. Often interesting interactions result that were not anticipated, suggesting ideas for further interactive neuroevolution of the team.

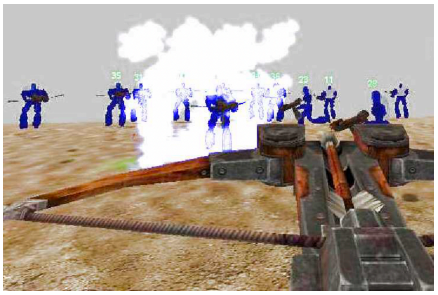
One of the first behaviors is often to approach a firing enemy. The agents quickly evolve to avoid fire by going around and approaching from the side. This behavior is general and adapts easily to enemies that are turning. If subsequently the "approach" slider is abruptly changed to "avoid" (i.e. negative rewards for approaching), and interesting demonstration of evolutionary search can be seen. As always, there are individuals in the population that do not perform very well. Even if most agents approach the enemy, some of them may stand still, roam around, or run away. When the slider changes, they become the seed for the behavioral change. They receive higher fitness, and their offspring takes over the population, resulting in avoidance in a few reproductions.

In some cases, careful curricular design can be used to construct effective desired behaviors. For instance it is possible to evolve the agents to run through a maze to a target on the other side. First the environment may consist of a single wall, and gradually more walls in complex configurations as the agents evolve to run around them (Figure 8.3*a*). The resulting behavior can be quite general and effective, despite involving no actual path planning. It is enough for the agents to know the general direction and they can navigate around even complex mazes, as long as they do not contain deceptive traps. Combined with the objective of dispersing, the agents also take different paths through the maze—which is effective because it is difficult to defend against an enemy that approaches from many directions at once.

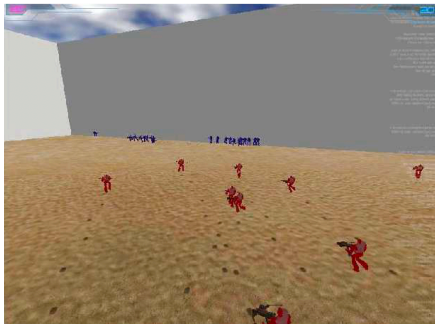
On the other hand, evolution can still discover surprising and effective behaviors as well. One such result was that the agents sometimes evolve to run backwards (Figure 8.3*b*). This seems odd at first, but does serve a purpose in some cases. If the enemy tends to pursue the agents persistently, running backwards is useful because the weapon remains pointed to the enemy. Another discovery was that extremely avoidant behavior can be effective in



(*a*) Running a maze



(*b*) Running backwards while shooting



(*c*) Forming a firing squad

(*b*) Subteams of three agents

Figure 8.3: **Discovery of expected and unexpected behaviors in NERO.** What makes the game interesting is that the player has some control of what will happen, but evolution will also find surprising solutions. (*a*) By gradually adding more walls, and rewarding the agents for staying away from each other, they evolve to take various paths through the maze, without any explicit path planning. (*b*) An effective strategy for hitting the target while not getting hit is to run backwards while shooting. (*c*) An avoidant team can be effective when they have time to back up against a wall, forming a firing squad. (*d*) A subteam of three agents is agile and has significant fire power. These discoveries and many more like them were surprising, resulting from evolution solving the challenges posed by the human player. In this manner, humans can provide guidance while still letting evolution to find creative solutions. For animations of these and other battle behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. (figures *a* – *c* from Stanley, Bryant, and Miikkulainen 2005)

battle (Figure 8.3*c*). That is, most of the time aggressive teams are evolved that approach the enemy and pursue it if they retreat. An avoidant team, however, would retreat until they have their back against the wall. It turns out if they are fast enough to do it so there is still enough of them, they form a firing squad that is very difficult to approach, and aggressive pursuers are often eliminated. Yet another surprising discovery was that some teams evolved to form subteams of three agents (Figure 8.3*d*): they approach the enemy together, they fire at the same enemy, and they retreat together. Such a subteam is effective because it has significant firepower yet is very agile. Evolution discovered it independently, however this principle turned out to be well established in actual military training.

One interesting question in NERO is: Is there an actual best strategy in the game, or does it support several different strategies that each dominate some, but not all, other strategies?

This is a crucial question for machine learning games in general, as well as interactive neuroevolution. While it is difficult to answer this question conclusively, it is possible to conduct a large scale experiment with many players and evaluate the resulting strategies.

The first MOOC on Artificial Intelligence in 2011, run by Peter Norvig and Sebastian Thrun, provided such an opportunity (Karpov, Johnson, and Miikkulainen 2015). As an optional assignment in the course, the students designed NERO teams, and a comprehensive round robit tournament was run with them. Out of the 156 submissions, some performed much better than others, and the teams could be ranked according to total wins: The best one won 137 times, then next 130, then two teams at 126, then 125, 124, 123 etc.

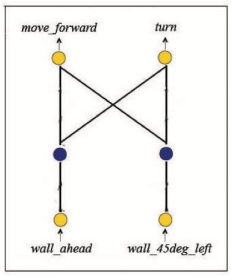
When the behavior was characterized in terms of actions taken in various situations, 10 major behavioral strategies were identified. However, none of them were clearly more successful than others; what mattered the most was how well they were implemented. What is most interesting, however, is that there was clear circularity among the best teams: Team A beat Team B which beat Team C which beat Team A. This result suggests that it is unlikely that one best strategy exists, but different behaviors are required to do well against different opponents. Both of these properties make the game more interesting to human players, and suggest that machine learning games is indeed a viable genre. They also suggest that human intuition in interactive evolution can be useful, and can provide an outlet for human creativity, as is also demonstrated in following sections of this chapter. Furthermore, combining human and machine insight is a powerful approach for designing complex systems.

The software for the original NERO as well as its open source version is available from the book website. The original NERO includes a version 2.0 of the game, which includes human guidance also during the battles, as well as an ability to construct teams by combining individuals from different evolutionary runs. The goal was to make the teams more versatile and the game-play more interactive; the interactive evolution aspect remained the same. OpenNERO was also designed to support other AI and machine learning methods, making it possible to compare and demonstrate different approaches to intelligent agents. They can serve as a starting point for exercises and projects in this book.

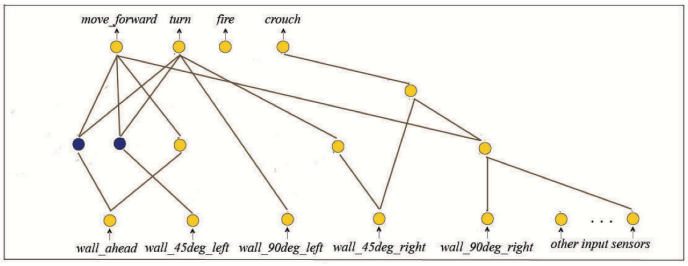
8.2 Incorporating Human Knowledge

NERO is one of the first examples of a genre of machine learning games, i.e. the gameplay consists of players interacting with a machine learning system. Its focus was on one particular kind of interaction, i.e. on shaping neuroevolution through human insight. However, it is possible to incorporate human knowledge into neuroevolution in other ways as well, including explicitly through rule-based advice and implicitly through behavioral examples.

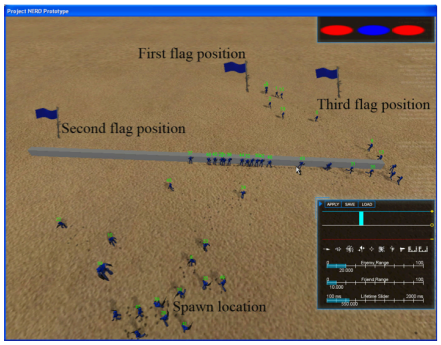
Note that these approaches are useful in creating intelligent agents in general; for instance, advice can be used in prey capture to help the agent evolve a corralling strategy, pushing the prey into the corner rather than chasing it in circles (Fan, Lau, and Miikkulainen 2003). Similarly, examples can be used to train agents in a strategy game to establish behavioral doctrines that also observe safety constraints, resulting in visibly intelligent behavior that does not easily emerge on its own in neuroevolution (Bryant and Miikkulainen 2007). However, advice and examples can be most clearly demonstrated and evaluated in NERO because it is an interactive evolution environment to begin with.



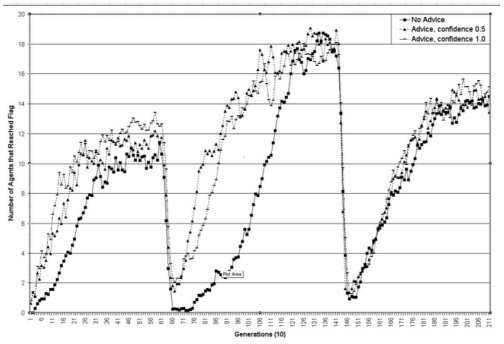
(*a*) The advice network structure



(*b*) Advice spliced into a NERO network



(*c*) The three phases of the experiment



(*d*) Performance over generations

Figure 8.4: **Utilizing rule-based advice in NERO.** It is sometimes useful to be able to guide the evolutionary discover with human knowledge. Such knowledge can be expressed as rules and incorporated into the population of networks.(*a*) As an example, two rules about going around the wall on the right side are encoded as a partial network structure. (*b*) This structure is then spliced into NEAT networks like any mutation. The networks continue to evolve to take advantage, modify, or co-opt the advice to perform better. (*c*) A snapshot of NERO with the three sequential positions identified. The agents were first rewarded for going to the flag in the middle, then to the one at left, then the one at right. (*d*) The advice suggested going to the first flag around the right side, and it sped up evolution compared to having no advice. When the flag was moved to the left, networks with advice adapted very quickly, utilizing the same advice structure with different output actions. After the flag was moved again, there was no difference in adaptation with or without advice, suggesting that the advice had become incorporated into the network like any other structure in it. (figures from Yong et al. 2006)

In NERO, successful behaviors are discovered through exploration. This means that even the most obvious ones, like moving around a wall without getting stuck, take many iterations of trial and error. This process is often frustrating to watch because effective behavior is obvious to the observer, and s/he might as well tell the agents what they should do. Evolution can then use that advice as a starting point, modify it further, and move on to more interesting discoveries faster.

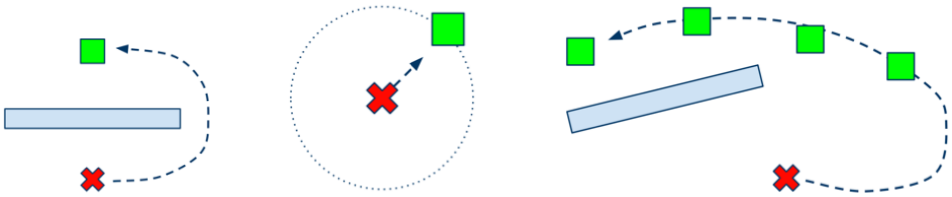
A mechanism for incorporating such advice into evolving neural networks can be built based on knowledge-based artificial neural networks (KBANN; Towell and Shavlik 1994).

The knowledge is first specified in a set of rules, such as "if a wall is some distance in front, then move forward and turn right" and "if a wall is near 45 degrees to the left, then move forward and turn slightly right." The rules are then converted into partial neural network structures: The conditions are coded as input nodes and consequents as output nodes, with hidden nodes mapping between them (Figure 8.4*a*,*b*; Yong et al. 2006). These structures are spliced into each existing neural network in the population, thus adding the wall-circling behavior to their existing behaviors. Weight values are usually constant, with a positive or negative sign, but can also be graded to indicate e.g. the degree of turn. Note that such additions are natural in NEAT, which already has mechanisms for growing the networks through add-node, add-connection, and change-weight mutations. Evolution then continues to modify these networks, incorporating the advice into the general behavior, modifying the advice to make it more useful, or even rejecting it entirely and changing it into something else. Confidence values can be used to specify how likely such modifications are, i.e. how immutable or plastic the advice is. Given that the evolutionary changes modify rules that were originally interpretable, the modifications may be interpretable as well, i.e. it may be possible to explain what new knowledge evolution discovers in this process.

Experiments demonstrate that such advice indeed helps learn the task of e.g. going around the wall faster (Figure 8.4*c*,*d*). Remarkably, if the task changes so that it is now better to go around the left side instead of the right, adaptation is very fast: evolution quickly changes the output actions to the left while the rest of the advice network structure stays the same. If the task changes again to make the right side better, there's little difference between networks that evolved with advice or not. In both cases, the advice has become incorporated into the general network structure. In this manner, advice helps evolution discover the needed behaviors but does not constrain evolution in the longer term.

In some cases, it may be difficult or inconvenient to write down advice as rules, but it may be easy to demonstrate the desired behavior by driving an agent in the game. For instance, the knowledge about going around a wall can be presented in this way. The agent is placed in a starting location, the player takes possession of it, and gives movement commands that take it to the target flag. At each step, the inputs and outputs to the agent are recorded and used as a training set with backpropagation through time; alternatively, the path of the agent can be divided into segments, and the actions that keep the agent on the example path used as targets. The agent is first trained to reproduce the first segment, then the first two, and so on until it successfully replicates the entire example. The weight changes are encoded back to the genetic encoding of the network (implementing Lamarckian evolution), and are thus inherited by its offspring.

It is interesting to evaluate how well each of these methods for incorporating human knowledge i.e. shaping, advice, and examples, work in interactive neuroevolution. To this end, a human-subject study was conducted (Karpov, Valsalam, and Miikkulainen 2011). A total of 16 participants were given three tasks: going around the wall, catching a moving target, and traversing a trajectory consisting of multiple waypoints (Figure 8.5). They were instructed to solve these tasks by two different methods: by writing a set of rules, i.e. a script for the entire behavior, and one other method which was either advice, examples, or shaping, randomly chosen and in random order. Their performance was recorded and they were surveyed afterward; the performance was also compared with plain neuroevolution from scratch without any human knowledge.



(*a*) Going around a wall (*b*) Catching a moving target (*c*) Traversing through waypoints

Figure 8.5: **Tasks for evaluating methods that incorporate human knowledge in NERO.** Plain neuroevolution from scratch on one hand and full scripting of behavior on the other were compared with advice, examples, and shaping. Plain neuroevolution turned out more successful than scripting, and at least one of the human-guided methods more successful than plain neuroevolution: examples in (*a*), advice in (*b*), and shaping in (*c*). Thus, the different methods of incorporating human knowledge can play a different role in constructing intelligent agents in interactive neuroevolution domains. (figures from Karpov, Valsalam, and Miikkulainen 2011)

The surveys suggested that the example-based approach was favored as the best quality approach, then scripting, shaping, and advice. Shaping was found to be low quality in the moving-target task, advice low quality in the waypoints task, and all methods were found good in the wall-circling task. These ratings did not always correlate with rate of success, suggesting that they mostly measure how easy or fun it was to use each method—which is useful information on its own.

The recordings were used to measure the average time to a successful solution, with a 30-minute upper bound. It turned out that scripting was the most difficult way to achieve successful performance: even plain neuroevolution was more successful. Interestingly, at least one human-assisted method performed better than plain neuroevolution. Advice was most effective in catching the moving target. It was possible to specify an intercept course rather than chasing the target indefinitely. In general, advice makes sense when the behavior can be expressed as a general rule. In contrast, examples were best in going-around-the-wall task. Indeed, this approach is most appropriate when the desired behavior is concrete and specific. Shaping, the usual staple of the NERO game, was the most effective in the waypoint task, where it was possible to start with a single target and then add gradually more waypoints. The approach makes sense in general in tasks where it is possible to start with a simplified or partial version and then gradually make the task more demanding. In this manner, each of the different ways of incorporating human knowledge into interactive neuroevolution can play a different role in constructing intelligent agents.

When exactly should each of these methods be used? An interesting possibility for the future is for the interactive evolution system itself to request advice, examples, and shaping when it deems it most helpful (Karpov et al. 2012). For instance, the system can identify parts of the state space where it has little experience, or that are least likely to lead to success, or where the population of agents disagrees the most, and where its previous advice or examples do not apply. It can then present the user with an advice template specifying such a situation and ask the user to fill in the blanks. Alternatively, it can present a starting point

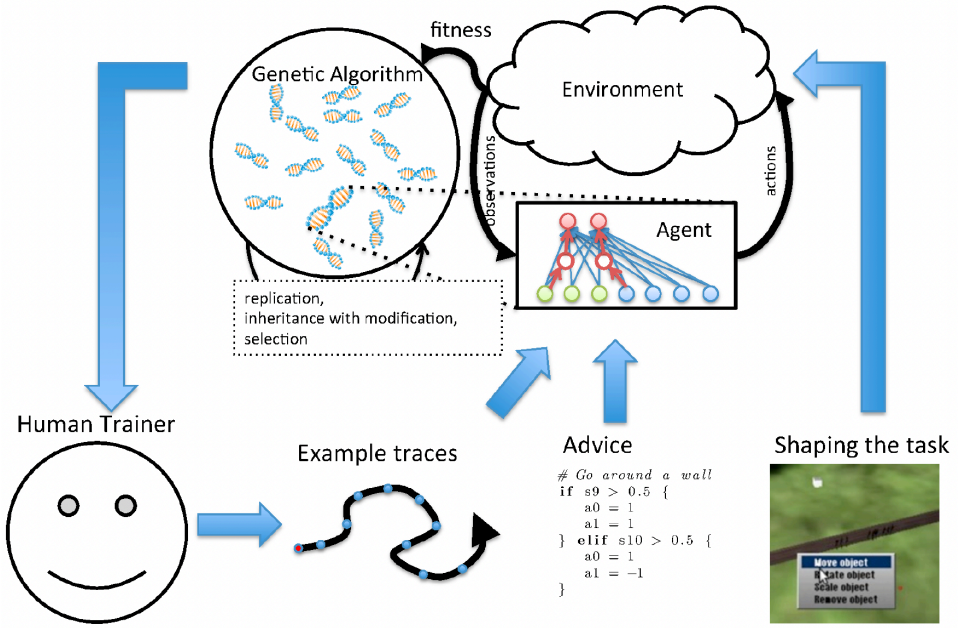


Figure 8.6: **A proposal for active human-guided neuroevolution.** The human expert provides advice, examples, and shaping for the neuroevolution process. The process monitors itself and determines what kind and when such input would be most useful. In this manner, humans and machines can work synergetically to construct intelligent agents. (Figure from Karpov et al. 2012)

for the agent and ask the user to provide an example. If evolution seems to have stagnated, it could propose the user to shape either the rewards or the environment to get evolution going again. It could even make specific suggestions, such as adjusting the sliders to make the task more demanding, or rolling back prior simplifications. Such an ability would eventually result in interactive neuroevolution where human knowledge and machine exploration work synergetically in both directions to solve problems (Figure 8.6).

8.3 Collaborative Neuroevolution

While NERO enabled players to shape the evolution of their team of agents, the game did not allow many humans to collaboratively train their teams by building on the interesting behaviors found by others. This chapter showcases some examples of interactive neuroevolution applications and games that were developed to incorporate such collaboration.

In particular, we'll take a closer look at Picbreeder (Secretan et al. 2011a), a highly influential generative AI system that came out of the lab of Kenneth Stanley. Picbreeder is a great example of a system that allows users to perform *collaborative* interactive neuroevolution, enabling them to explore a large design space together. Similarly to Dawkin's BioMorphs from his book The Blind Watchmaker, the basic idea in Picbreeder is to breed images. Users are presented with several images and asked to select the ones they like the most (Figure 8.7). The selected images are then used as parents to produce a new generation