Video games provide an ideal platform for the development and testing of machine-learning techniques.

Creating Intelligent Agents in Games

Risto Miikkulainen is a professor of computer sciences at the University of Texas at Austin.

Risto Miikkulainen

Games have long been a popular area for research in artificial intelligence (AI), and for good reason. Because games are challenging yet easy to formalize, they can be used as platforms for the development of new AI methods and for measuring how well they work. In addition, games can demonstrate that machines are capable of behavior generally thought to require intelligence without putting human lives or property at risk.

Most AI research so far has focused on games that can be described in a compact form using symbolic representations, such as board games and card games. The so-called good old-fashioned artificial intelligence (GOFAI; Haugeland, 1985) techniques work well with symbolic games, and to a large extent, GOFAI techniques were developed for them. GOFAI techniques have led to remarkable successes, such as Chinook, a checkers program that became the world champion in 1994 (Schaeffer, 1997), and Deep Blue, the chess program that defeated the world champion in 1997 and drew significant attention to AI in general (Campbell et al., 2002).

Since the 1990s, the field of gaming has changed tremendously. Inexpensive yet powerful computer hardware has made it possible to simulate complex physical environments, resulting in tremendous growth in the video game industry. From modest sales in the 1960s (Baer, 2005), sales of entertainment software reached $25.4 billion worldwide in 2004 (Crandall and Sidak, 2006). Video games are now a regular part of many people's lives, and the market continues to expand.

Curiously, very little AI research has been involved in this expansion. Many video games do not use AI techniques, and those that do are usually based on relatively standard, labor-intensive scripting and authoring methods. In this and other respects, video games differ markedly from symbolic games. Video games often involve many agents embedded in a simulated physical environment where they interact through sensors and effectors that take on numerical rather than symbolic values. To be effective, agents must integrate noisy input from many sensors, react quickly, and change their behavior during the game. The AI techniques developed for and with symbolic games are not well suited to video games.

In contrast, machine-learning techniques, such as neural networks, evolutionary computing, and reinforcement learning, are very well suited to video games. Machine-learning techniques excel in exactly the kinds of fast, noisy, numerical, statistical, and changing domains that today's video games provide. Therefore, just as symbolic games provided an opportunity for the development and testing of GOFAI techniques in the 1980s and 1990s, video games provide an opportunity for the development and testing of machine-learning techniques and their transfer to industry.

Very little machine learning is used in current commercial video games.

Artificial Intelligence in Video Games

One of the main challenges for AI is creating intelligent agents that can become more proficient in their tasks over time and adapt to new situations as they occur. These abilities are crucial for robots deployed in human environments, as well as for various software agents that live in the Internet or serve as human assistants or collaborators.

Although current technology is still not sufficiently robust to deploy such systems in the real world, they are already feasible in video games. Modern video games provide complex artificial environments that can be controlled and carry less risk to human life than any real-world application (Laird and van Lent, 2000). At the same time, video gaming is an important human activity that occupies millions of people for countless hours. Machine learning can make video games more interesting and reduce their production costs (Fogel et al., 2004) and, in the long run, might also make it possible to train humans realistically in simulated, adaptive environments. Video gaming is, therefore, an important application of AI and an excellent platform for research in intelligent, adaptive agents.

Current video games include a variety of high-realism simulations of human-level control tasks, such as navigation, combat, and team and individual tactics and strategy. Some of these simulations involve traditional AI techniques, such as scripts, rules, and planning (Agre and Chapman, 1987; Maudlin et al., 1984), and a large part of AI development is devoted to path-finding algorithms, such as A*-search and simple behaviors built using finite-state machines. AI is used to control the behavior of the non-player characters (NPCs, i.e., autonomous computer-controlled agents) in the game. The behaviors of NPCs, although sometimes impressive, are often repetitive and inflexible. Indeed, a large part of the gameplay in many games is figuring out what the AI is programmed to do and learning to defeat it.

Machine learning in games began with Samuel's (1959) checkers program, which was based on a method similar to temporal-difference learning (Sutton, 1988). This was followed by various learning methods applied to tic-tac-toe, backgammon, go, Othello, and checkers (see Fürnkranz, 2001, for a survey). Recently, machine-learning techniques have begun to appear in video games as well. For example, Fogel et al. (2004) trained teams of tanks and robots to fight each other using a competitive coevolution system, and Spronck (2005) trained agents in a computer role-playing game using dynamic scripting. Others have trained agents to fight in first- and third-person shooter games (Cole et al., 2004; Hong and Cho, 2004). Machine-learning techniques have also been applied to other video game genres, from Pac-Man (Lucas, 2005) to strategy games (Bryant and Miikkulainen, 2003; Yannakakis et al., 2004).

Nevertheless, very little machine learning is used in current commercial video games. One reason may be that video games have been so successful that a new technology such as machine learning, which would fundamentally change the gaming experience, may be perceived as a risky investment by the industry. In addition, commercial video games are significantly more challenging than the games used in research so

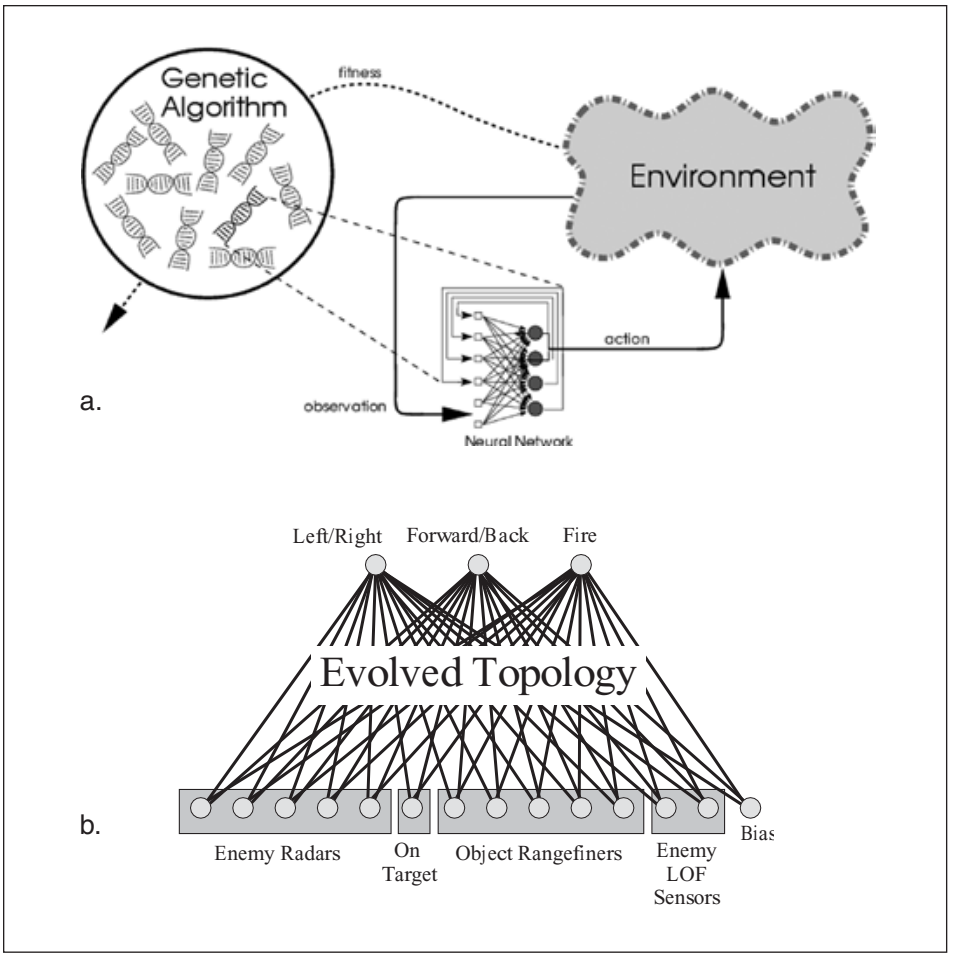

FIGURE 1 1a. Evolving neural networks. Solutions (such as neural networks) are encoded as chromosomes, usually consisting of strings of real numbers, in a population. Each individual is evaluated and assigned a fitness based on how well it performs a given task. Individuals with high fitness reproduce; individuals with low fitness are thrown away. Eventually, nearly all individuals can perform the task. 1b. Each agent in neuroevolution receives sensor readings as input and generates actions as output. In the NERO video game, the network can see enemies, determine whether an enemy is currently in its line of fire, detect objects and walls, and see the direction the enemy is firing. Its outputs specify the direction of movement and whether or not to fire. In this way, the agent is embedded in its environment and must develop sophisticated behaviors to do well. For general neuroevolution software and demos, see *http://nn.cs.utexas.edu.*

far. They not only have large state and action spaces, but they also require diverse behaviors, consistent individual behaviors, fast learning, and memory of past situations (Gomez et al., 2006; Stanley et al., 2005)

Neuroevolution

The rest of this article is focused on a particular machine-learning technique, neuroevolution, or the evolution of neural networks. This technique not only promises to rise to the challenge of creating games that are educational, but also promises to provide a platform for the safe, effective study of how intelligent agents adapt.

Evolutionary computation is a computational machine-learning technique modeled after natural evolution (Figure 1a). A population of candidate solutions are encoded as strings of numbers. Each solution is evaluated in the task and assigned a fitness based on how well it performs. Individuals with high fitness are then reproduced (by crossing over their encodings) and mutated (by randomly changing components of their encodings with a low probability). The offspring of the high-fitness individuals replace the low-fitness individuals in the population, and over time, solutions that can solve the task are discovered.

In neuroevolution, evolutionary computation is used to evolve neural network weights and structures. Neural networks perform statistical pattern transformation and generalization, and evolutionary adaptation allows for learning without explicit targets, even with little reinforcement. Neuroevolution is particularly well suited to video games because (1) it works well in high-dimensional spaces; (2) diverse populations can be maintained; (3) individual networks behave consistently; (4) adaptation takes place in real time; and (5) memory can be implemented through recurrency (Gomez et al., 2006; Stanley et al., 2005).

Several methods have been developed for evolving neural networks (Yao, 1999). One particularly appropriate for video games is called neuroevolution of augmenting topologies (NEAT; Stanley and Miikkulainen, 2002), which was originally developed for learning behavioral strategies. The neural networks control agents that select actions in their output based

on sensory inputs (Figure 1b). NEAT is unique in that it begins evolution with a population of small, simple networks and *complexifies* those networks over generations, leading to increasingly sophisticated behaviors.

With neuroevolution, entirely new game genres can be developed.

NEAT is based on three key ideas. First, for neural network structures to increase in complexity over generations, a method must be found for keeping track of which gene is which. Otherwise, it will not be clear in later generations which individuals are compatible or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique historical marking to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited unchanged during crossover, which allows NEAT to perform crossover without expensive topological analysis. Thus, genomes of different organizations and sizes remain compatible throughout evolution.

Second, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. In this way, topological innovations are protected and have time to optimize their structures. NEAT uses the historical markings on genes to determine the species to which different individuals belong.

Third, unlike other systems that evolve network topologies and weights, NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem. This process of complexification has important implications for the search for solutions. Although it may not be practical to find a solution in a high-dimensional space by searching that space directly, it may be possible to find it by first searching in lower dimensional spaces and complexifying the best solutions into the high-dimensional space.

As is usual in evolutionary algorithms, the entire population is replaced with each generation in NEAT. However, in a real-time game or simulation, this would seem incongruous because every agent's behavior would change at the same time. In addition, behaviors would remain static during the large gaps between generations. Therefore, in order to apply NEAT to video games, a real-time version of it, called rtNEAT, was created.

In rtNEAT, a single individual is replaced every few game ticks. One of the poorest performing individuals is removed and replaced with a child of parents chosen from among the best performing individuals. This cycle of removal and replacement happens continually throughout the game and is largely invisible to the player. As a result, the algorithm can evolve increasingly complex neural networks fast enough for a user to interact with evolution as it happens in real time. This real-time learning makes it possible to build machine-learning games.

Machine-Learning Games

The most immediate opportunity for neuroevolution in video games is to build a "mod," a new feature or extension, to an existing game. For example, a character that is scripted in the original game can be turned into an adapting agent that gradually learns and improves as the game goes on. Or, an entirely new dimension can be added to the game, such as an intelligent assistant or tool that changes as the player progresses through the game. Such mods can make the game more interesting and fun to play. At the same time, they are easy and safe to implement from a business point of view because they do not change the original structure of the game. From the research point of view, ideas about embedded agents, adaptation, and interaction can be tested with mods in a rich, realistic game environment.

With neuroevolution, however, learning can be taken well beyond game mods. Entirely new game genres can be developed, such as machine-learning games, in which the player explicitly trains game agents to perform various tasks. The fun and challenge of machine-learning games is to figure out how to take agents through successive challenges so that in the end they perform well in their chosen tasks. Games such as Tamagotchi "Virtual Pet" and Black & White "God Game" suggest that

interaction with artificial agents can make for viable and entertaining games. In NERO, the third such game, the artificial agents adapt their behavior through sophisticated machine learning.

The NERO Game

The main idea of NERO is to put the player in the role of a trainer or drill instructor who teaches a team of agents by designing a curriculum. The agents are simulated robots that learn through rtNEAT, and the goal is to train them for military combat.

The agents begin the game with no skills but with the ability to learn. To prepare them for combat, the player must design a sequence of training exercises and goals. Ideally, the exercises will be increasingly difficult so that the team begins by learning basic skills and then gradually builds on them (Figure 2). When the player is satisfied that the team is well prepared, the team is deployed in a battle against another team trained by another player, allowing the players to see if their training strategies pay off.

The challenge is to anticipate the kinds of skills that might be necessary for battle and build training exercises to hone those skills. A player sets up training exercises by placing objects on the field and specifying goals through several sliders. The objects include static enemies, enemy turrets, rovers (i.e., turrets that move), flags, and walls. To the player, the sliders serve as an interface for describing ideal behavior. To rtNEAT, they represent coefficients for fitness components. For example, the sliders specify how much to reward or punish agents for approaching enemies, hitting targets, getting hit, following friends, dispersing, etc. Each individual fitness component is normalized to a Z-score (i.e., the number of standard deviations from the mean) so all components can be measured on the same scale. Fitness is computed as the sum of all components multiplied by their slider levels, which can be positive or negative. Thus, the player has a natural interface for setting up a training exercise and specifying desired behavior.

Agents have several types of sensors (Figure 1b). Although NERO programmers frequently experiment with new sensor configurations, the standard sensors include enemy radars, an "on target" sensor, object range finders, and line-of-fire sensors. To ensure consistent evaluations, agents all begin in a designated area of the field called the factory. Each agent is allowed to spend a limited amount of time on the field during which its fitness can be assessed. When time on the field expires, the agent is transported back to the factory, where another evaluation begins.

Training begins by deploying 50 agents on the field. Each agent is controlled by a neural network with random connection weights and no hidden nodes, which is the usual starting configuration for NEAT. As the neural networks are replaced in real-time, behavior improves, and agents eventually learn to perform the task the player has set up. When the player decides performance has reached a satisfactory level, he or she can save the team in a file. Saved teams can be reloaded for further training in different scenarios, or they can be loaded into battle mode.

In battle mode, the player discovers how well the training has worked. Each player assembles a battle team of 20 agents from as many different trained teams as desired, possibly combining agents with different skills.

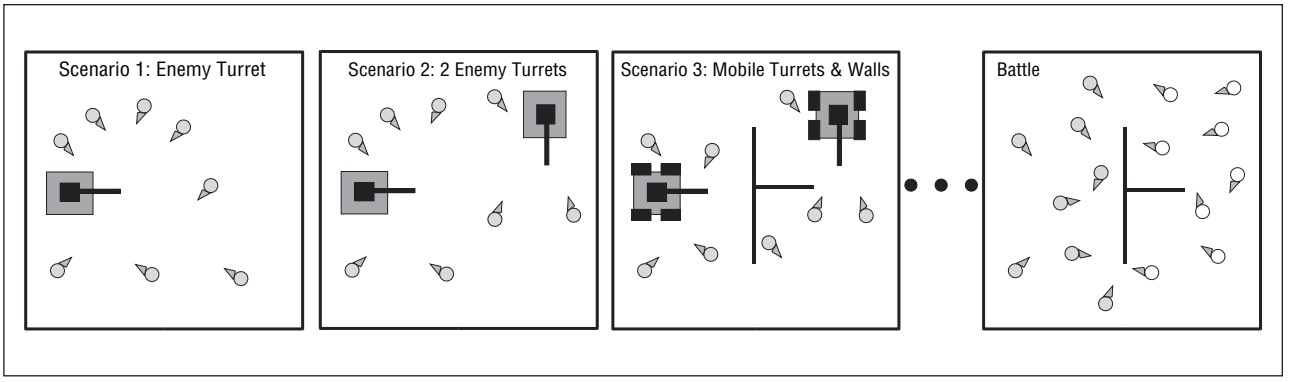

FIGURE 2 A sample training sequence in NERO. The figure depicts a sequence of increasingly difficult training exercises in which agents attempt to attack turrets without getting hit. In the first exercise, there is only a single turret; additional turrets are added by the player as the team improves. Eventually walls are added, and the turrets are given wheels so they can move. Finally, after the team has mastered the hardest exercises, it is deployed in a battle against another team. For animations of various training and battle scenarios, see *http://nerogame.org*.

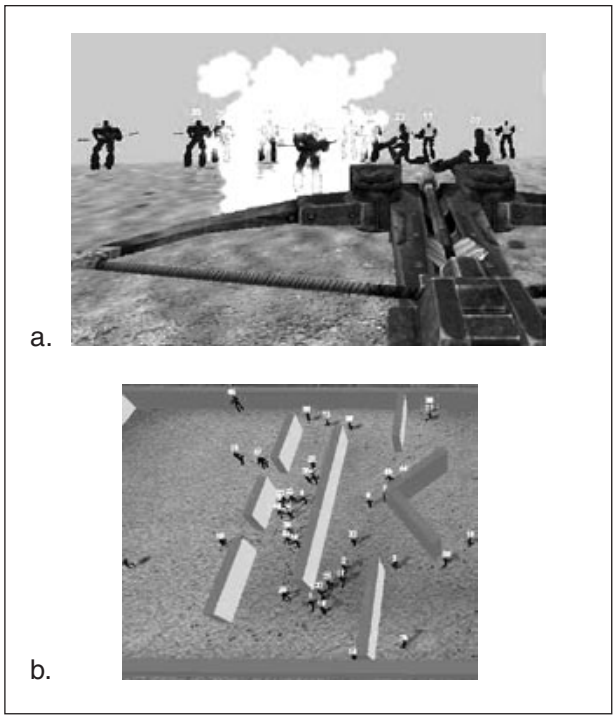

FIGURE 3 Behaviors evolved in NERO. 3a. This training screenshot shows several agents running away backward while shooting at the enemy, which is being controlled from a first-person perspective by a human trainer with a joystick. This scenario demonstrates how evolution can discover novel and effective behaviors in response to challenges set up by the player. 3b. Incremental training on increasingly complex wall configurations produced agents that could navigate this complex maze to find the enemy. Remarkably, they had not seen this maze during training, suggesting that they had evolved general path-navigation ability. The agents spawn from the left side of the maze and proceed to an enemy at the right. Notice that some agents evolved to take the path through the top, while others evolved to take the bottom path, suggesting that protecting innovation in rtNEAT supports a range of diverse behaviors with different network topologies. Animations of these and other behaviors can be seen at *http://nerogame.org*.

The battle begins with two teams arrayed on opposite sides of the field. When one player presses a "go" button, the neural networks take control of their agents and perform according to their training. Unlike training, however, where being shot does not cause damage to an agent's body, agents in battle are destroyed after being shot several times (currently five). The battle ends when one team is completely eliminated. In some cases, the surviving agents may insist on avoiding each other, in which case the winner is the team with the most agents left standing.

Torque, a game engine licensed from GarageGames (*http://www.garagegames.com/*), drives NERO's simulated physics and graphics. An important property of Torque is that its physics is slightly nondeterministic so that the same game is never played twice. In addition, Torque makes it possible for the player to take control of enemy robots using a joystick, an option that can be useful in training.

Behavior can be evolved very quickly in NERO, fast enough so that the player can be watching and interacting with the system in real time. The most basic battle tactic is to seek the enemy aggressively and fire at it. To train for this tactic, a single static enemy is placed on the training field, and agents are rewarded for approaching the enemy. This training requires that agents learn to run toward a target, which is difficult because they start out in the factory facing in random directions. Starting with random neural networks, it takes on average 99.7 seconds for 90 percent of the agents on the field to learn to approach the enemy successfully (10 runs, sd = 44.5s).

Note that NERO differs from most applications of evolutionary algorithms in that the quality of evolution is judged from the player's perspective based on the performance of the entire population, instead of the performance of the population champion. However, even though the entire population must solve the task, it does not converge to the same solution. In seek training, some agents evolve a tendency to run slightly to the left of the target, while others run to the right. The population diverges because the 50 agents interact as they move simultaneously on the field at the same time. If all of the agents chose exactly the same path, they would often crash into each other and slow each other down, so agents naturally take slightly different paths to the goal. In other words, NERO is a massively parallel, coevolving ecology in which the entire population is evaluated together.

Agents can also be trained to avoid the enemy, leading to different battle tactics. In fact, rtNEAT is flexible enough to devolve a population that has converged on seeking behavior into its complete opposite, a population that exhibits avoidance behavior. For avoidance training, players control an enemy robot with a joystick and run it toward the agents on the field. The agents learn to back away to avoid being penalized for being too near the enemy. Interestingly, they prefer to run away from the enemy backward so they can still see and shoot at the enemy (Figure 3a). As an interesting combination of conflicting goals, a turret can be placed on the field and agents asked to approach it without getting hit. As a result, they learn to avoid enemy fire, running to the side opposite the bullets and approaching the turret from behind. This tactic is also effective in battle.

Other interesting behaviors have been evolved to test the limits of rtNEAT, rather than specifically prepare troops for battle. For example, agents were trained to run around walls in order to approach the enemy. As performance improved, players incrementally added more walls until the agents could navigate an entire maze (Figure 3b). This behavior was remarkable because it was successful without any path planning.

The agents developed the general strategy of following any wall that stood between them and the enemy until they found an opening. Interestingly, different species evolved to take different paths through the maze, showing that topology and function are correlated in rtNEAT and confirming the success of real-time speciation. The evolved strategies were also general enough for agents to navigate significantly different mazes without further training. In another example, when agents that had been trained to approach a designated location (marked by a flag) through a hallway were attacked by an enemy controlled by the player, they learned, after two minutes, to take an alternative path through an adjacent hallway to avoid the enemy's fire. Such a response is a powerful demonstration of real-time adaptation. The same kind of adaptation could be used in any interactive game to make it more realistic and interesting.

Teams that were trained differently were sometimes surprisingly evenly matched. For example, a seeking team won six out of ten battles, only a slight advantage, against an avoidant team that ran in a pack to a corner of the field next to an enclosing wall. Sometimes, if an avoidant team made it to the corner and assembled fast enough, the seeking team ran into an ambush and was obliterated. However, slightly more often the seeking team got a few shots in before the avoidant team could gather in the corner. In that case, the seeking team trapped the avoidant team and had more surviving numbers. Overall, neither seeking nor avoiding provided a significant advantage.

Strategies can be refined further by observing behaviors during battle and setting up training exercises to improve them. For example, a seeking team could eventually be made more effective against an avoidant team when it was trained with a turret that had its back against the wall. The team learned to hover near the turret and fire when it turned away and to back off quickly when it turned toward them. In this way, rtNEAT can discover sophisticated tactics that dominate over simpler ones. The challenge for the player is to figure out how to set up the training curriculum so sophisticated tactics will emerge.

NERO was created over a period of about two years by a team of more than 30 student volunteers (Gold, 2005). The game was first released in June 2005 at *http://nerogame.org* and has since been downloaded more than 100,000 times. NERO is under continuing development and is currently focused on providing more interactive play. In general, players agree that the game is engrossing and entertaining. Battles are exciting, and players spend many hours perfecting behaviors and assembling teams with just the right combination of tactics. Remarkably, players who have little technical background often develop accurate intuitions about the underlying mechanics of machine learning. This suggests that NERO and other machine-learning games are viable as a genre and may even attract a future generation of researchers to machine learning.

Games with adapting intelligent agents are likely to be in high demand.

Games like NERO can be used as research platforms for implementing novel machine-learning techniques. For example, one direction for research is to incorporate human knowledge, in terms of rules, into evolution. This knowledge could then be used to seed the population with desired initial behaviors or to give real-time advice to agents during evolution (Cornelius et al., 2006; Yong et al., 2006). Another area for research is to learn behaviors that not only solve a given problem, but solve it in a way that makes sense to a human observer. Although such solutions are difficult to describe formally, a human player may be able to demonstrate them by playing the game himself or herself. An evolutionary learning system can then use these examples to bias learning toward similar behaviors (Bryant, 2006).

Conclusion

Neuroevolution is a promising new technology that is particularly well suited to video game applications. Although neuroevolution methods are still being developed, the technology can already be used to make current games more challenging and interesting and to

implement entirely new genres of games. Such games, with adapting intelligent agents, are likely to be in high demand in the future. Neuroevolution may also make it possible to build effective training games, that is, games that adapt as the trainee's performance improves.

At the same time, video games provide interesting, concrete challenges for machine learning. For example, they can provide a platform for the systematic study of methods of control, coordination, decision making, and optimization, within uncertainty, material, and time constraints. These techniques should be widely applicable in other fields, such as robotics, resource optimization, and intelligent assistants. Just as traditional symbolic games catalyzed the development of GOFAI techniques, video gaming may catalyze research in machine learning for decades to come.

Acknowledgments

This work was supported in part by the Digital Media Collaboratory of the University of Texas at Austin, Texas Higher Education Coordinating Board through grant ARP-003658-476-2001, and National Science Foundation through grants EIA-0303609 and IIS-0083776.

References

Agre, P.E., and D. Chapman. 1987. Pengi: An Implementation of a Theory of Activity. Pp. 268–272 in Proceedings of the 6th National Conference on Artificial Intelligence. Los Altos, Calif.: Morgan Kaufmann.

Baer, R.H. 2005. Videogames: In the Beginning. Springfield, N.J.: Rolenta Press.

Bryant, B.D. 2006. Evolving Visibly Intelligent Behavior for Embedded Game Agents. Ph.D. thesis, University of Texas at Austin. Technical Report AI-06-334.

Bryant, B.D., and R. Miikkulainen. 2003. Neuroevolution for Adaptive Teams. Pp. 2194–2201 in Proceedings of the 2003 Congress on Evolutionary Computation. Piscataway, N.J.: IEEE.

Campbell, M., A.J. Hoane Jr., and F.-H. Hsu. 2002. Deep Blue. Artificial Intelligence 134: 57–83.

Cole, N., S.J. Louis, and C. Miles. 2004. Using a Genetic Algorithm to Tune First-Person Shooter Bots. Pp. 139–145 in Proceedings of the 2004 Congress on Evolutionary Computation. Piscataway, N.J.: IEEE.

Cornelius, R., K.O. Stanley, and R. Miikkulainen. 2006. Constructing Adaptive AI Using Knowledge-Based Neuroevolution. Pp. 693–708 in AI Game Programming Wisdom 3, edited by S. Rabin. Revere, Mass.: Charles River Media.

Crandall, R.W., and J.G. Sidak. 2006. Video Games: Serious Business for America's Economy. Entertainment Software Association Report. Available online at: *http://www.theesa.com/files/VideoGame_Final.pdf*.

Fogel, D.B., T.J. Hays, and D.R. Johnson. 2004. A Platform for Evolving Characters in Competitive Games. Pp. 1420–1426 in Proceedings of the 2004 Congress on Evolutionary Computation. Piscataway, N.J.: IEEE.

Fürnkranz, J. 2001. Machine Learning in Games: A Survey. Pp. 11–59 in Machines That Learn to Play Games, edited by J. Fürnkranz and M. Kubat. Huntington, N.Y.: Nova Science Publishers.

Gold, A. 2005. Academic AI and Video Games: A Case Study of Incorporating Innovative Academic Research into a Video Game Prototype. Pp. 141–148 in Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games. Piscataway, N.J.: IEEE.

Gomez, F., J. Schmidhuber, and R. Miikkulainen. 2006. Efficient Non-linear Control through Neuroevolution. Pp. 654–662 in Proceedings of the European Conference on Machine Learning. Berlin: Springer-Verlag.

Haugeland, J. 1985. Artificial Intelligence: The Very Idea. Cambridge, Mass.: MIT Press.

Hong, J.-H., and S.-B. Cho. 2004. Evolution of Emergent Behaviors for Shooting Game Characters in Robocode. Pp. 634–638 in Proceedings of the 2004 Congress on Evolutionary Computation. Piscataway, N.J.: IEEE.

Laird, J.E., and M. van Lent. 2000. Human-Level AI's Killer Application: Interactive Computer Games. Pp. 1171–1178 in Proceedings of the 17th National Conference on Artificial Intelligence. Menlo Park, Calif.: AAAI Press.

Lucas, S.M. 2005. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. Pp. 203–210 in Proceedings of the IEEE Symposium on Computational Intelligence and Games. Piscataway, N.J.: IEEE.

Maudlin, M.L., G. Jacobson, A. Appel, and L. Hamey. 1984. ROG-O-MATIC: A Belligerent Expert System. In Proceedings of the 5th National Conference of the Canadian Society for Computational Studies of Intelligence. Mississagua, Ontario: Canadian Society for Computational Studies of Intelligence.

Samuel, A.L. 1959. Some studies in machine learning using the game of checkers. IBM Journal 3: 210–229.

Schaeffer, J. 1997. One Jump Ahead. Berlin: Springer-Verlag.

Spronck, P. 2005. Adaptive Game AI. Ph.D. thesis, Maastricht University, the Netherlands.

Stanley, K.O., B.D. Bryant, and R. Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation 9(6): 653–668.

Stanley, K.O., and R. Miikkulainen. 2002. Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2): 99–127.

Sutton, R.S. 1988. Learning to predict by the methods of temporal differences. Machine Learning 3: 9–44.

Yannakakis, G.N., J. Levine, and J. Hallam. 2004. An Evolutionary Approach for Interactive Computer Games. Pp. 986–993 in Proceedings of the 2004 Congress on Evolutionary Computation. Piscataway, N.J.: IEEE.

Yao, X. 1999. Evolving artificial neural networks. Proceedings of the IEEE 87(9): 1423–1447.

Yong, C.H., K.O. Stanley, R. Miikkulainen, and I. Karpov. 2006. Incorporating Advice into Evolution of Neural Networks. In Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference. Menlo Park, Calif.: AAAI Press.