Creating Intelligent Agents in Games

Risto Miikkulainen
The University of Texas at Austin

Abstract

Game playing has long been a central topic in artificial intelligence. Whereas early research focused on utilizing search and logic in board games, machine learning in video games is driving much of the recent research. In video games, intelligent behavior can be naturally captured through interaction with the environment, and biologically inspired techniques such as evolutionary computation, neural networks, and reinforcement learning are well suited for this task. In particular, neuroevolution, i.e. constructing artificial neural network agents through simulated evolution, has shown much promise in many game domains. Based on sparse feedback, complex behaviors can be discovered for single agents and for teams of agents, even in real time. Such techniques may allow building entirely new genres of video games that are more engaging and entertaining than current games, and can even serve as training environments for people. Techniques developed in such games may also be widely applicable to other fields, such as robotics, resource optimization, and intelligent assistants.

1 Introduction

Games have long been a popular area of artificial intelligence (AI) research, and for a good reason. They are challenging yet easy to formalize, making it possible to develop new AI methods, measure how well they are working, and demonstrate that machines are capable of impressive behavior generally thought to require intelligence without putting human lives or property at risk.

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

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 an explosion of the video game industry. From modest beginnings in the 1960s (Baer 2005), the entertainment software sales have expanded to $25.4 billion worldwide in 2004 (Crandall and Sidak 2006). Video games have become a facet of many people's lives and the market continues to expand.

Curiously, this expansion has involved little AI research. Many video games utilize no AI techniques, and those that do are usually based on relatively standard, labor-intensive scripting and authoring methods. The reason is that video games are very different from the symbolic games. There are often many agents involved, embedded in a simulated physical environment where they interact through sensors and effectors that take on numerical rather than symbolic values. To be effective, the agents have to integrate noisy input from many sensors, and they have to react quickly and change their behavior during the game. The techniques that have been developed for and with symbolic games are not well suited for video games.

In contrast, machine learning (ML) techniques such as neural networks, evolutionary computing, and reinforcement learning are very well suited for video games. They excel in exactly the kinds of fast, noisy, numerical, statistical, and changing domains that today's video games provide. Therefore, video games constitute an opportunity similar to that of the symbolic games for GOFAI in 1980s and 1990s: an opportunity to develop and test ML techniques, and an opportunity to transfer the technology to industry.

2 AI in Video Games

One of the main challenges for AI is to create intelligent agents that adapt, i.e. change their behavior based on interactions with the environment, becoming more proficient in their tasks over time, and adapting to new situations as they occur. Such ability is crucial for deploying robots in human environments, as well as for various software agents that live in the Internet or serve as human assistants or collaborators.

While general such systems are still beyond current technology, they are already possible in special cases. In particular, video games provide complex artificial environments that can be controlled, and carry perhaps the least risk to human life of any real-world application (Laird and van Lent 2000). On the other hand, such games are an important part of human activity, with millions of people spending countless hours on them. Machine learning can make video games more interesting and decrease their production costs (Fogel et al. 2004). In the long run, such technology might also make it possible to train humans realistically in simulated adaptive environments. Video gaming is therefore an important application of AI on its own right, 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, 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). A large fraction of AI development is devoted to path-finding algorithms such as A*-search and simple behaviors built using finite state machines. The AI is used to control the behavior of the non-player-characters (NPCs), i.e. the autonomous computer-controlled agents in the game. Although such agents can exhibit impressive behaviors, they 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.

More recently, machine-learning techniques have begun to appear in video games. This trend follows a long history of learning in board games, originating from Samuel's (1959) checkers program that was based on a method similar to temporal difference learning (Sutton 1988), followed by various learning methods applied to tic-tac-toe, backgammon, go, othello, and checkers (see Fürnkranz 2001 for a survey). Many of these learning methods can be applied to 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). ML 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).

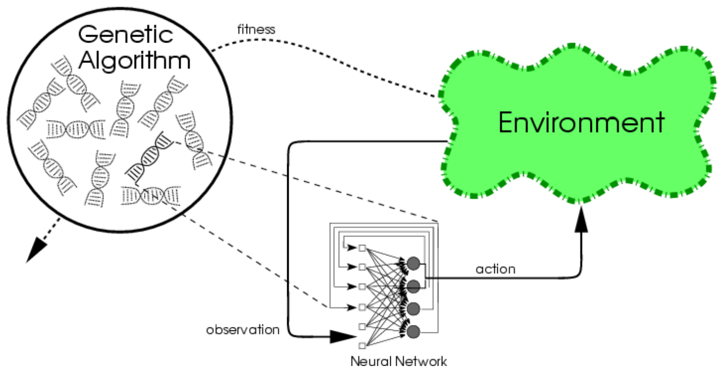
Still, there is very little ML in current commercial video games. One reason may be that video games have been too successful; a new technology such as ML, which would fundamentally change the gaming experience, is perceived as a risky investment by the industry. Seond, modern video games are significantly more challenging than the games used in research so far. They have large state and action spaces, and require diverse behaviors, consistent individual behaviors, fast learning, and memory of past situations (Stanley et al. 2005).

This paper presents one particular ML technique that promises to rise up to the challenge: evolution of neural networks, or neuroevolution. It also reviews one example of how this technique can be successful in building machine learning games. Such games can be entertaining, but they can also be educational. In the long term, they provide a research platform in which adapting intelligent agents can be studied safely and effectively.

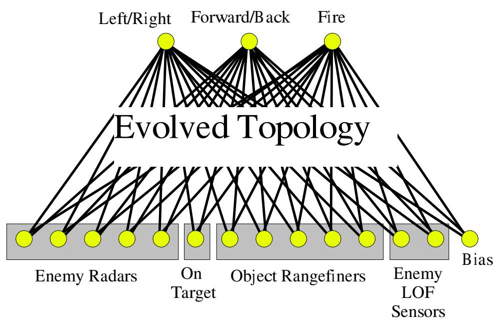
3 Neuroevolution

Evolutionary computation (EC) is a computational learning technique modeled after natural evolution (Figure 1a). A population of candidate solutions are encoded as strings of numbers. Each one 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 replaces the low-fitness individuals in the population, and over time, solutions that can solve the task are discovered.

In neuroevolution, EC is used to evolve neural network weights and structures. Neural networks perform statistical pattern transformation and generalization, and evolutionary adaptation allows learning the networks without explicit targets, even with sparse reinforcement. The approach is particularly well-suited for video games: NE works well in high-dimensional spaces, diverse populations can be maintained, individual networks behave consistently, adaptation takes place in real time, and memory can be implemented through recurrency (Gomez et al. 2006; Stanley et al. 2005).



(*a*) Evolutionary Learning



(*b*) A Neural Network Agent

Figure 1: **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 in the task and assigned a fitness based on how well it performs. Individual with high fitness reproduce while those with low fitness are thrown away, eventually resulting in individuals that can solve the task. (b) Each agent in neuroevolution receives sensor readings as its input and generates actions as its output. In the NERO video game (Section 4, 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 manner, the agent is embedded in its environment and needs to develop sophisticated behaviors to do well. For neuroevolution software and demos, see `http://nn.cs.utexas.edu`.

Several methods exist for evolving neural networks (Yao 1999). One particularly appropriate for video games is NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen 2002), originally developed for learning behavioral strategies. The neural networks control agents that select actions in their output based on their sensory inputs (Figure 1*b*). NEAT is unique in that it begins evolution with a population of small, simple networks and *complexify* those networks over generations, leading to increasingly sophisticated behaviors.

NEAT is based on three key ideas. First, in order to allow neural network structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which, 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 during crossover unchanged, and allow NEAT to perform crossover without the need for expensive topological analysis. That way, genomes of different organizations and sizes stay compatible throughout evolution. Second, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before competing with other niches in the population. NEAT uses the historical markings on genes to determine to which species 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 search: While it may not be practical to find a solution in a high-dimensional space by searching in 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 at each generation in NEAT. However, in a real time game or a simulation, such a step would look incongruous since every agent's behavior would change at once. 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 worst individuals is removed and replaced with a child of parents chosen from among the best. This cycle of removal and replacement happens continually throughout the game and is largely invisible to the player. The result is an algorithm that can evolve increasingly complex neural networks fast enough for a user to interact with evolution as it happens in real time. Such real-time learning makes it possible to build machine-learning games, as will be described next.

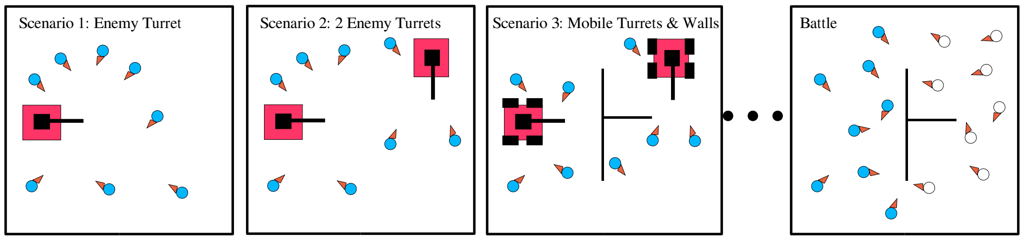

Scenario 1: Enemy Turret
Scenario 2: 2 Enemy Turrets
Scenario 3: Mobile Turrets & Walls
Battle

Figure 2: **A Sample Training Sequence in NERO.** The figure depicts a sequence of increasingly difficult training exercises in which the agents attempt to attack turrets without getting hit. In the first exercise there is only a single turret but more 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 real battle against another team. For animations of various training and battle scenarios, see `http://nerogame.org`.

4 Building Machine Learning Games

The most immediate opportunity for NE in video games is to build a "mod", or 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, gradually learning and improving as the game goes on. Or, and entirely new dimension can be added to the game, such as an intelligent assistant or a tool, that changes as the player progresses through the game. Such mods can make the game more interesting and fun to play; on the other hand, they are easy and safe to implement since they do not change the original structure of the game. From the research point of view, mods allow testing ideas about embedded agents, adaptation, and interaction in a rich and realistic game environment.

However, with current NE techniques it is possible to take learning well beyond game mods, and develop entirely new game genres. One such genre is Machine Learning Games, where the player explicitly trains game agents to perform various tasks. The fun and the challenge of such games is to figure out how to take the agents through successive challenges so that in the end they perform well in the chosen tasks. Games such as Tamagotchi virtual pet and Black & White "god game" suggest that such interaction with artificial agents can make a viable and entertaining game. This section describes how advanced learning methods such as NE can be utilized to build complex machine learning games. As an example, the NERO game (Stanley et al. 2005) is built based on the rtNEAT method.

The main idea of NERO is to put the player in the role of a trainer or a drill instructor who teaches a team of agents by designing a curriculum. The agents are simulated robots, and the goal is to train a team of these agents for military combat. The agents begin the game with no skills and only the ability to learn. In order to prepare for combat, the player must design a sequence of training exercises and goals. Ideally, the exercises are increasingly difficult so that the team can begin by learning basic skills and then gradually build 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 player to see how his or her training strategy paid off. The challenge is to anticipate the kinds of skills that might be necessary for battle and build training exercises to hone those skills.

The 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 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 that they are measured on the same scale. Fitness is computed as the sum of all these 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 1*b*). Although NERO programmers frequently experiment with new sensor configurations, the standard sensors include enemy radars, an "on target" sensor, object rangefinders, and line-of-fire sensors. To ensure consistent evaluations, the agents spawn from a designated area of the field called the factory. Each agent is allowed a limited time on the field during which its fitness is assessed. When their time on the field expires, agents are transported back to the factory, where they begin another evaluation.

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 sets up. When the player decides that 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 worked out. Each player assembles a battle team of 20 agents from as many different trained teams as desired, possibly combining teams with different skills. The battle begins with the two teams arrayed on opposite sides of the field. When one player presses a "go" button, the neural networks obtain control of their agents and perform according to their training. Unlike in training, where being shot does not lead to an agent body being damaged, the agents are actually destroyed after being shot several times (currently five) in battle. The battle ends when one team is completely eliminated. In some cases, the only surviving agents may insist on avoiding each other, in which case the winner is the team with the most agents left standing.

The game engine Torque, licensed from GarageGames (`http://www.garagegames.com/`), drives NERO's simulated physics and graphics. An important property of the Torque engine 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.

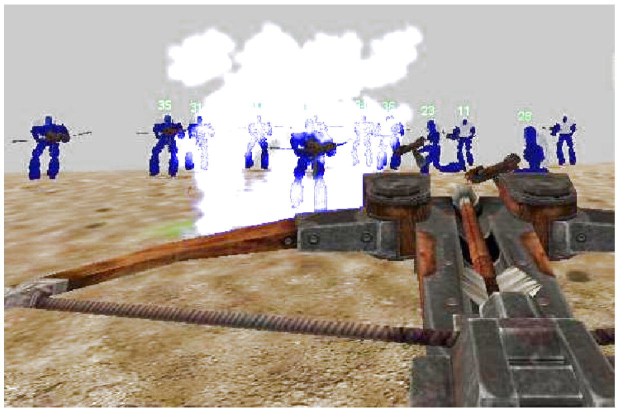
Behavior can be evolved very quickly in NERO, fast enough so that the player can be watching and interacting with the system in real time. The most basic battle tactic is to aggressively seek the enemy and fire. To train for this tactic, a single static enemy was placed on the training field, and agents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

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 that 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 evolved 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 the agents chose exactly the same path, they would often crash into each other and slow each other down, so naturally some agents take slightly different paths to the goal. In other words, NERO is actually a massively parallel coevolving ecology in which the entire population is evaluated together.

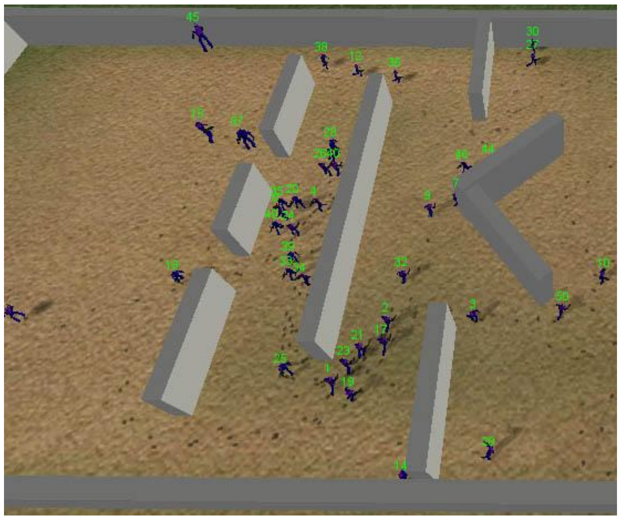
Agents were also trained to avoid the enemy. In fact, rtNEAT was flexible enough to *devolve* a population that had converged on seeking behavior into a completely opposite, avoidance, behavior. For avoidance training, players controlled an enemy robot with a joystick and ran it towards the agents on the field. The agents learned to back away in order to avoid being penalized for being too near the enemy. Interestingly, they preferred to run away from the enemy backwards, because that way they could still see and shoot at the enemy (figure $3a$). Further, by placing a turret on the field and asking agents to approach it without getting hit, they were able to learn to avoid enemy fire. They run to the side opposite of the bullets and approach the turret from behind, a tactic that is also effective in battle.

Other interesting behaviors were evolved to test the limits of rtNEAT, rather than specifically prepare the 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 is remarkable because it is successful without any path planning. The agents developed the general strategy of following any wall that stands 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 to navigate significantly different mazes without further training. Alo, in a powerful demonstration of real-time adaptation, agents that were trained to approach a designated location (marked by a flag) through a hallway were then attacked by an enemy controlled by the player. After two minutes, the agents learned to take an alternative path through an adjacent hallway in order to avoid the enemy's fire. While such training is used in NERO to prepare agents for battle, the same kind of adaptation could be used in any interactive game to make it more realistic and interesting.

In battle, some teams that were trained differently were nevertheless evenly matched, while some training types consistently prevailed against others. For example, an aggressive team had only a slight advantage over an avoidant team, winning six out of ten battles. The avoidant team runs in a pack to a corner of the field's enclosing wall.



(a) Avoiding the enemy effectively



(b) Navigating a maze

Figure 3: **Behaviors Evolved in NERO.** (a) This training screenshot shows several agents running away backwards and shooting at the enemy, which is being controlled from a first-person perspective by a human trainer with a joystick. Agents discovered this behavior during avoidance training because it allows them to shoot as they flee. This result demonstrates how evolution can discover novel and effective behaviors in response to the tasks that the player sets up for them. (b) 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 a general path-navigation ability was evolved. 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. This result suggests that protecting innovation in rtNEAT supports a range of diverse behaviors, each with its own network topology. Animations of these and other behaviors can be seen at `http://nerogame.org`.

Sometimes, if they make it to the corner and assemble fast enough, the aggressive team runs into an ambush and is obliterated. However, slightly more often the aggressive team gets a few shots in before the avoidant team can gather in the corner. In that case, the aggressive team traps the avoidant team with greater surviving numbers. The conclusion is that seeking and avoiding are fairly well-balanced tactics, neither providing a significant advantage over the other.

Strategies can be refined further by observing the behaviors in the battle, and setting up training exercises to improve them. For example, the aggressive team could eventually be made much more effective against the avoidant team by training them with a turret with its back against the wall. This team learned to hover near the turret and fire when it turned away, but back off quickly when it turned towards them. In this manner, 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 that they will emerge.

NERO was created over a period of about two years by a team of over 30 student volunteers (Gold 2005). It was first released in June of 2005 at `http://nerogame.org`, and has since then been downloaded over 60,000 times. It is under continuing development, currently focused on providing more interactive gameplay. In general, players agree that the game is engrossing and entertaining. Battles are exciting events, and players spent many hours honing 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 experience is promising, suggesting that NERO and other machine learning games are viable as a genre, and may even serve to attract a future generation of researchers to machine learning.

Machine learning games such as NERO can be used as a research platform for implementing novel machine learning techniques. For example, one important direction is to incorporate human knowledge, in terms of rules, into evolution: Such knowledge can be used to seed the population with desired initial behaviors, or it can be used to give real-time advice to agents during evolution (Cornelius et al. 2006; Yong et al. 2006). Another one is to learn behaviors that not only solve the problem, but do 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 towards similar behaviors (Bryant 2006).

5 Conclusion

Neuroevolution is a promising new technology that is particularly well suited for video game applications. Although NE methods are still being developed, the technology can already be used to make current games more challenging and interesting, and to implement entirely new game genres. Such games, with adapting intelligent agents, are likely to be in high demand in the future. In addition, they may finally make it possible to build effective training games, i.e. those that adapt as the trainee gets better. At the same time, video games provide interesting, concrete challenges for machine learning. For example, methods for control, coordination, decision making, and optimization, with uncertainty, material, and time constraints, can be studied systematically in such games. The techniques should be widely applicable in other fields, such as robotics, resource optimization, and intelligent assistants. Like traditional symbolic games for GOFAI, video gaming may thus serve as a catalyst for research in machine learning for decades to come.

Acknowledgments

This work was supported in part by the DMC Lab of the Univ. of Texas at Austin, by the Texas Higher Education Coordinating Board through grant ARP-003658-476-2001, and by NSF through grants EIA-0303609 and IIS-0083776.

References

Agre, P. E. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, volume 1, pages 268–272, Los Altos, CA. Morgan Kaufmann.

Baer, R. H. (2005). *Videogames: In the Beginning*. Rolenta Press, Springfield, NJ.

Bryant, B. D. (2006). *Evolving Visibly Intelligent Behavior for Embedded Game Agents*. PhD thesis, The Univ. of Texas at Austin.

Bryant, B. D. and Miikkulainen, R. (2003). Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 3, pages 2194–2201, Piscataway, NJ. IEEE.

Campbell, M., Hoane Jr., A. J., and hsiung Hsu, F. (2002). Deep Blue. *Artificial Intelligence*, 134:57–83.

Cole, N., Louis, S. J., and Miles, C. (2004). Using a genetic algorithm to tune first-person shooter bots. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, pages 139–145, Piscataway, NJ. IEEE.

Cornelius, R., Stanley, K. O., and Miikkulainen, R. (2006). Constructing adaptive AI using knowledge-based neuroevolution. In Rabin, S., editor, *AI Game Programming Wisdom 3*. Charles River Media.

Crandall, R. W. and Sidak, J. G. (2006). Video games: Serious business for America's economy. Entertainment Software Assoc.

Fogel, D. B., Hays, T. J., and Johnson, D. R. (2004). A platform for evolving characters in competitive games. In *Proceedings of 2004 Congress on Evolutionary Computation*, pages 1420–1426, Piscataway, NJ. IEEE Press.

Fürnkranz, J. (2001). Machine learning in games: A survey. In Fürnkranz, J. and Kubat, M., editors, *Machines That Learn to Play Games*, chapter 2, pages 11–59. Nova Science Publishers, Huntington, NY.

Gold, A. (2005). Academic AI and video games: A case study of incorporating innovative academic research into a video game prototype. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2006). Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*.

Haugeland, J. (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA.

Hong, J.-H. and Cho, S.-B. (2004). Evolution of emergent behaviors for shooting game characters in robocode. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 634–638, Piscataway, NJ. IEEE.

Laird, J. E. and van Lent, M. (2000). Human-level AI's killer application: Interactive computer games. In *Proceedings of the 17th National Conference on Artificial Intelligence*, Menlo Park, CA. AAAI Press.

Lucas, S. M. (2005). Evolving a neural network location evaluator to play ms. pac-man. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, Piscataway, NJ. IEEE.

Maudlin, M. L., Jacobson, G., Appel, A., and ard Hamey, L. (1984). ROG-O-MATIC: A belligerent expert system. In *Proceedings of the Fifth National Conference of the 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*. Springer, Berlin.

Spronck, P. (2005). *Adaptive Game AI*. PhD thesis, Maastricht University, the Netherlands.

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

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10:99–127.

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

Yannakakis, G. N., Levine, J., and Hallam, J. (2004). An evolutionary approach for interactive computer games. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 986–993, Piscataway, NJ. IEEE.

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

Yong, C. H., Stanley, K. O., Miikkulainen, R., and Karpov, I. (2006). Incorporating advice into evolution of neural networks. In *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*.