Integration and Evaluation of Exploration-Based Learning in Games

Igor V. Karpov[1], **Thomas D'Silva**[2], **Craig Varrichio**[3], **Kenneth O. Stanley**[4], **Risto Miikkulainen**[1]
[1],[2],[3] Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, USA
[4] School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA

.

Abstract

Video and computer games provide a rich platform for testing adaptive decision systems such as value-based reinforcement learning and neuroevolution. However, integrating such systems into the game environment and evaluating their performance in it is time and labor intensive. In this paper, an approach is developed for using general integration and evaluation software to alleviate these problems. In particular, the Testbed for Integrating and Evaluating Learning Techniques (TIELT) is used to integrate a neuroevolution learner with an off-the-shelf computer game Unreal Tournament$^{\text{TM}}$[5] (Aha and Molineaux 2004). The resulting system is successfully used to evolve artificial neural network controllers with basic navigation behavior. Our work leads to formulating a set of requirements that make a general integration and evaluation system such as TIELT a useful tool for benchmarking adaptive decision systems.

1 Introduction

It is often necessary to compare new machine learning techniques empirically against other similar techniques to gauge how they improve on previous results. In order to minimize the influence of *data bias* in the evaluation process, new algorithms should be tested in as large a variety of domains as possible, especially when general claims about their performance are being made. Existing video and computer games provide a rich platform for testing adaptive decision systems such as value-based reinforcement learning and neuroevolution. However, integrating and evaluating multiple algorithms and implementations against multiple simulation domains is a difficult process. Moreover, any benchmarking results can also be skewed by *implementation bias*, or intentional or unintentional disparity in quality of implementation of the proposed approach vs. the quality of implementation of competing approaches (Keogh and Kassetty 2002).

One potential way to make empirical comparisons of learning techniques easier and more principled is to build a flexible software platform that provides a uniform interface for the learning system across multiple rich simulation domains. Such a system allows researchers to concentrate on implementing and optimizing their approach to solving the learning problem and to validate it empirically by comparing with other approaches.

In this paper we present the results of using one such "glue" framework to integrate an existing reinforcement learning technique with an existing game environment. The "glue" framework used is the Testbed for Integrating and Evaluating Learning Techniques (TIELT), developed by David Aha and Matthew Molineaux (Aha and Molineaux 2004). It has been designed to "integrate AI systems with (e.g., real-time) gaming simulators, and to evaluate how well those systems learn on selected simulation tasks" (Molineaux and Aha 2005).

The project uses NeuroEvolution of Augmenting Topologies (NEAT) method as the learning technique (Stanley and Miikkulainen 2002a). NEAT is a genetic algorithm that evolves increasingly complex artificial neural network controllers by applying mutation and crossover operators to their populations based on a fitness function. A real time variant of NEAT, called rtNEAT, has been used successfully in the game NERO, produced by the Digital Media Collaboratory at the University of Texas at Austin (Gold 2005; Stanley et al. 2005b).

Using TIELT, NEAT is embedded into Unreal Tournament$^{\text{TM}}$, a popular First Person Shooter (FPS) computer game produced by Epic Games, Inc. in 1999 and winning a Game of the Year title for that year. Unreal Tournament$^{\text{TM}}$was previously integrated with the TIELT system and other learning methods were tested on it. However, exploration-based learning algorithms such as value-function and evolution-based reinforcement learning have not been tested with Unreal/TIELT before (Molineaux 2004).

The next section describes the state of AI in gaming, the learning technique of neuroevolution, and the TIELT approach to integration and evaluation. Section 3 gives an overview of our system's architecture, Section 4 summarizes experiments performed and their results, and Section 5 analyses the results and describes future work.

2 Background

2.1 AI in gaming

Video game technology can provide a rich platform for validating and advancing theoretical AI research (Gold 2005). On the other hand, the video game industry stands to benefit from the adapting artificial intelligence and machine learning

[1]{ikarpov,risto}@cs.utexas.edu
[2]tdsilva@mail.utexas.edu
[3]cavarrichio@yahoo.com
[4]kstanley@cs.ucf.edu
[5]Unreal Tournament is a trademark of Epic Games, Inc.

techniques into new, more interactive games.

Computer and video games constitute a large and lucrative market, 7.3 billion dollars in the US in 2004 (ESA 2005). This allows game studios to develop a variety of what can be seen as high realism simulations for human-level control tasks, such as navigation, combat, team and individual tactics and strategy. The emergence of online multi-player and massively multi-player games offers unprecedented opportunities for real-time interaction between human players and artificially intelligent elements. Because of these factors, computer and video games may indeed be a ’killer application’ for developing and validating machine learning techniques (Laird and van Lent 2000).

However, adaptive algorithms are used in the game industry setting surprisingly rarely. Most popular video games on the market today use scripted non-player characters to add interactivity to their games (Gold 2005). A large fraction of AI development in the industry is devoted to path-finding algorithms such as A*-search and simple behaviors built using finite state machines. There may be several reasons for why this is the case. One is that high-end game applications have traditionally pushed hardware systems to their limits and simply did not have the resources to perform online learning. This is becoming less of a problem with the availability of cheap processing power and true parallelism. Another, deeper reason may be that it is difficult to adopt results from academic AI to games because they are not built and tested with these applications directly in mind. Where the goal of an AI researcher is often to build a system that is able to adapt to an environment to solve difficult tasks, the goal of a game developer is to build a system that is “human-like” and “fun”. Integration systems such as TIELT can allow researchers and engineers to more easily integrate and test new learning algorithms with games, benefiting both academic AI research and commercial game development.

2.2 Neuroevolution

Neuroevolution is a powerful technique for solving non-linear, non-Markovian control tasks (Gomez 2003). In neuroevolution, a genetic algorithm is used to evolve artificial neural networks. NeuroEvolution of Augmenting Topologies (NEAT) is a particularly efficient method of this kind able to evolve both network weights and topologies (Stanley and Miikkulainen 2002b). NEAT starts with a population of minimal network topologies and complexifies them when necessary to solve the problem at hand. NEAT is able to protect innovation through a speciation mechanism, and has an effective encoding scheme that allows it to perform mutation and recombination operations efficiently.

Neuroevolution and NEAT in particular have been shown to outperform other methods on a number of benchmark learning tasks. For example, neuroevolution has been successfully used in a number of game-playing domains (Agogino et al. 2000; Stanley et al. 2005a). It is therefore important to show that methods such as NEAT can benefit from general integration and evaluation tools such as TIELT.

2.3 Testbed for Integration and Evaluation of Learning Techniques (TIELT)

The Testbed for Integration and Evaluation of Learning Techniques, or TIELT, is a Java application intended to connect a game engine to a decision system that learns about the game (Aha and Molineaux 2004). The goal of the system is to provide the researcher with a tool that simplifies the task of testing a general learner in several different environments.

The application consists of five *knowledge bases* or modules which correspond to different areas of TIELT’s functionality. The *Game Model* encapsulates the information about the game from a single player’s perspective. The *Game Interface Model* describes how TIELT communicates with the game. The *Decision System Interface Model* describes interactions with the decision system. The *Agent Description* describes tasks performed by the system in order to play the game, and the *Experimental Methodology* module is used as an evaluation platform for particular games and learning techniques.

During integration, TIELT is connected with the game environment and with the learning system using the appropriate knowledge bases. The software allows the user to define the communication protocols between TIELT, the game engine and the learning system. TIELT also includes a visual scripting language that allows scripting of the game behavior and update rules.

TIELT provides the ability to connect environments and learners with arbitrary interfaces and rules into a cohesive learning system and to automate the evaluation of this system’s performance. This paper explores how well these features support neuroevolution.

3 System architecture

At the highest level, the system consists of three parts: the Unreal Tournament$^{\text{TM}}$server, the TIELT integration platform, and the decision system based on NEAT. The game server simulates the environment for the learning agent. TIELT communicates with the environment and accumulates a state, which is then communicated to the decision system. The decision system selects an action in the environment and communicates it back through TIELT. The decision system continually adopts its behavior by evolving artificial neural network controllers to perform the task.

3.1 Game engine

Unreal Tournament$^{\text{TM}}$is a real-time first person shooter (FPS) computer game in which a player navigates a three-dimensional space. The player can walk, run, jump, turn, shoot and interact with objects in the game such as armor, doors and health vials. The goal of one variant of the game (the tournament) is to be the first to destroy a certain number of opponents. Up to 16 players can play the game concurrently in a single level. A player can be controlled either by a human player connected over a network or an automated bot controlled by a built-in script.

The Gamebots API from the University of Southern California modifies the original game to allow players to be controlled via sockets connected to other programs such as an adaptive decision system (Kaminka et al. 2002). The communication protocol consists of synchronous and asynchronous sensory messages sent from the server to all the clients and of commands sent from the clients back to the server. The server has all the information about player locations, interactions and status. The synchronous updates occur about every 100 milliseconds, and include updates to the player's view of the game state. Asynchronous events include collisions with objects, players and projectiles, results of queries by the player and game over events.

3.2 Integration System

TIELT communicates with the GameBots API using TCP/IP sockets. The game interface model defines a subset of the GameBots protocol which is used to update the game model and trigger agent actions.

The Game Model represents the knowledge that a player has about the game at any given time. It includes the locations and types of objects encountered in the game, the objects that are currently visible or reachable, the location and heading of players, health, armor and other player status indicators. Additionally, the game model holds an array of eight Boolean variables that correspond to whether the locations distributed at a fixed radius around the player's most recent position are reachable. This game model allows the information from synchronous and asynchronous updates to be combined into a single state that can then be used by the decision system to generate appropriate actions. Because TIELT scripting language did not implement the operations necessary to combine these values into useful sensory inputs, the final values were calculated in our decision system implementation.

The Decision System Interface Model uses Java Reflection to dynamically load and use libraries of Java classes. These classes implement the NEAT learning system, as described in more detail in the next subsection.

The Agent Description is a script that sends sensor information from the Game Model to the Decision System and executes the resulting action on each synchronous update from the Unreal Tournament^{TM}server. This process is performed many times to evaluate a single individual.

The Experimental Methodology module in TIELT allows the user to specify which Game Model, Decision System, and Agent Description are to be used in an experiment and how many repetitions of the experiment have to be run. In our system, a single TIELT experiment corresponds to the evaluation of a single individual's fitness, and must be repeated $e \times p$ times, where e is the number of epochs and p is the population size. The state of the NEAT algorithm is persisted in memory across TIELT experiments.

3.3 Decision System

The output of the decision system is controlled by evolving neural networks. Each static neural network in a population performs a number of decisions during a predefined lifetime of the individual in the Unreal game environment. The resulting behavior is analyzed to compute a fitness score (section 4). The Unreal Tournament^{TM}world is then reset and a new network is used for decision-making. The decision system keeps track of individuals, their fitness functions, evaluation times, populations and epochs. The resulting algorithm is thus simply a serial variant of the neuroevolution algorithm, which evaluates each individual in turn (Algorithm 3.3).

Algorithm 1 Agent description tasks executed by TIELT

```
$Population \leftarrow RandomPopulation()$
for each epoch $e$ do
  for each individual $I$ do
    while time not expired do
      $s \leftarrow SensorValues()$ // TIELT to NEAT
      $a \leftarrow Action(I, s)$ // NEAT to TIELT
      $Act(a)$ // TIELT to Unreal
    end while
    $Fitness(I, e) \leftarrow$ C - MinDistanceToTarget
  end for
  $Population \leftarrow$ NextPopulation(P,Fitness)
end for
```

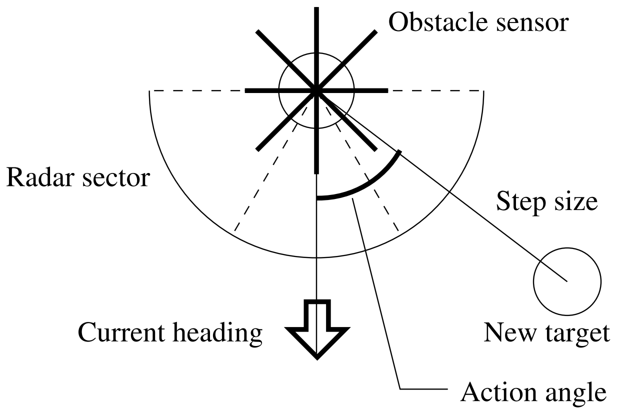

Figure 1: **Agent sensors and actions.** Each neural network has 11 sensory inputs: 8 radial Boolean obstacle sensors and 3 forward-facing "radar" sensors, whose values are derived from the distance to navigation points visible in each of the three sectors shown on the figure.

Sensors: Each neural network has 11 egocentric sensor inputs: eight boolean obstacle sensors $S_0..S_7$ with a small radius (in relative scale of the game they are roughly equivalent to stretching out the bot's arms in 8 directions) and three forward-directed 60-degree "radar" values $R_0..R_2$ (Figure 1). In addition to these 11 sensors, each network has a bias input (a constant 0.3 value found appropriate in previous NEAT experiments). Each radar value R_I is computed as $R_I = \sum_{x \in N_I} d(x)/C$ where N_I is the collection of navigation landmarks visible in sector I, $d(x)$ is the distance to each landmark, and C is a constant scaling factor. Thus, the R_I can be interpreted as the amount of free space in the direction; with higher activations corresponding to more visible navigation landmarks and to landmarks that are visible further away (Figure 1).

Actions: At each update, the decision system scales the single output of the network to a relative yaw angle $\Delta\alpha$ in the range of $[-\pi, \pi]$. TIELT then sends a command to the Unreal game server to move towards a point $(x + s\cos(\alpha + \Delta\alpha), y + s\sin(\alpha + \Delta\alpha), z)$ where s is the step size parameter.

4 Experiments

A number of validation experiments were conducted to verify that the learning system design is effective. The testing was done on an Intel Pentium 4 machine with a 2.4GHz clock rate and 1GB of RAM running Microsoft Windows XP. TIELT version 0.7 alpha and a Java implementation of NEAT were running on Sun Java Runtime Environment 1.5. An off-the-shelf copy of Unreal Tournament Game of the Year Edition was used in "dedicated server" mode (graphics disabled) in conjunction with a June 8, 2001 build of GameBots API. The following parameters were used:

- **Number of epochs:** 100 generations were used due to time constraints. This number of epochs was sufficient to evolve networks that were able to approach a static target.

- **Population size and target speciation:** The population size was set to 50, 100 and 200 individuals with the target speciation of 5, 10 and 20 species, respectively.

- **Number of repeated evaluations** In order to improve controller robustness in the presence of latency and noise, individual evaluations were repeated 1, 3 and 5 times and the average was used as the fitness function.

- **Evaluation time:** Each individual was evaluated for 10, 20 and 30 seconds, which translates to about 100, 200 and 300 consecutive actions.

- **Fitness function:** Throughout the lifetime of an individual, the system tracks the minimum distance d_{min} to a static target. The fitness function f for the individual is computed as $D - d_{min}$ where D is a constant greater than the largest measurement of the game level such that $f \geq 0$.

The task in the experiment was to navigate through the environment to a static target (Figure 2). At the beginning of an evaluation, the bot is placed at the origin and performs actions for the duration of the evaluation. The minimal distance d_{min} to the target is measured over the synchronous updates received by the learning system. The fitness function $C - d_{min}$ grows to a maximum value of C when the bot is able to approach the target.

In our initial experiments, we are able to reliably evolve controllers for the simple "go to target" task (Figure 3). Starting from initially random behavior, the record population fitness improved from 3080 to 3350 in 20 generations. At that time the best agent was able to navigate reliably to within 650 distance units, and further evolution produced better controllers.

Additional proof-of-concept experiments were performed on a Condor cluster of Intel Pentium 4 (2.4GHz) machines running Debian Linux. The results of these experiments were consistent with the data presented here, and demonstrate that it is possible to use a clustered environment to speed up evaluation of learning methods with TIELT.

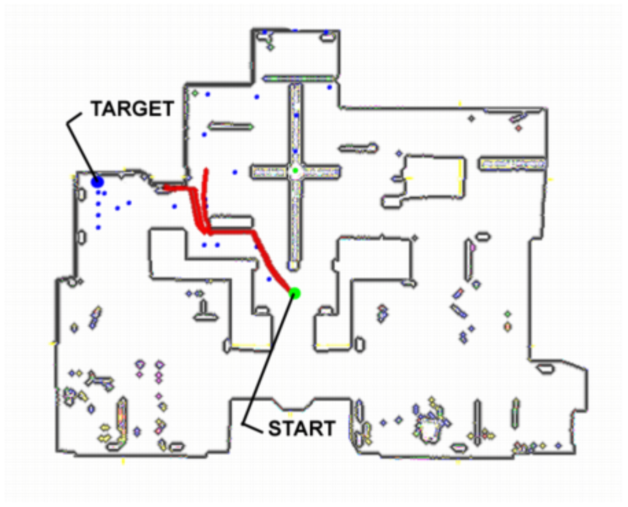

Figure 2: **Example of a path to target task.** Three traces of the best neural network navigating from the starting position to the ending position in an Unreal Tournament^{TM}level. The network shown is the result of 33 generations of evolution and has the fitness score of 3518 out of 4000.

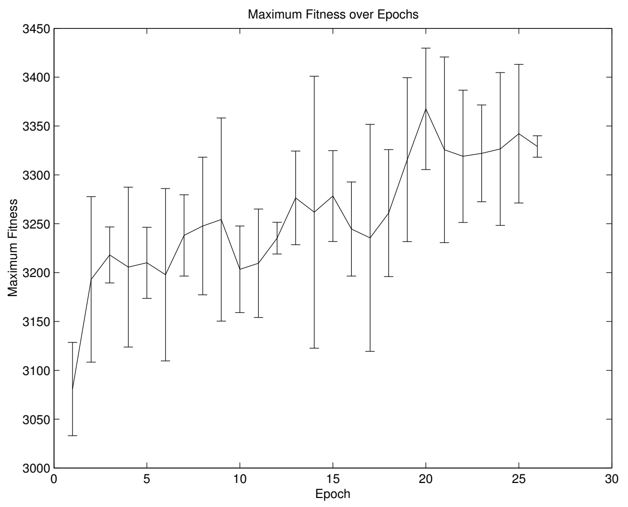

Figure 3: **Average population fitness with epochs.** Average fitness of 100 individuals per epoch, calculated as a mean of 3 10-second evaluations of the target navigation task in Figure 2. The values shown are averages over 6 26-epoch runs with the standard deviations shown by error bars.

5 Evaluation and Future Work

One important result of our work is an estimate of the time and effort required when using TIELT to integrate and evaluate a new decision system and the additional specifications for the future evolution of such systems. The project discussed in this paper consumed three academic hours of time for one undergraduate and two graduate students during the course of a semester, with regular consultation by the developer of TIELT and by the authors of NEAT. TIELT made the work of applying neuroevolution to Unreal Tournament simpler in several ways. Above all, the communication layer between Unreal and TIELT was already implemented and required only minor adjustments to integrate with the NEAT decision system.

At the end of the semester, the learner-environment system had basic functionality to evolve agents as single actors in the environment. However, the system was found to not be well-suited for running and analyzing repeated experiments efficiently. TIELT's design did not provide a convenient way to parallelize evaluations of agents or to evolve multiple agents acting in the same environment. As a result, a large portion of the functionality that can be taken care of in the "glue" framework has to be implemented in the decision system. This makes the decision system specific to the Unreal Tournament application and reduces the advantages of the middle layer.

There are several ways in which the TIELT system could be modified or other similar integration platforms be built in the future to better support exploration-based learning. In particular, the system can be more efficient, provide better support for batch experiments, easier to use, more flexible and more open.

Efficiency: Because the integration, evaluation and decision systems are implemented as Java applications, they can incur unexpected garbage collection delays and are often slower than native implementations. The Unreal Tournament server, which executes separately and as a native binary, does not incur such delays. In addition, the extra layers of indirection between the environment and the decision system add computational overhead to the process of making an individual decision. We observed these factors add up to create highly variable per-action latency, which introduces new challenges into learning a task in a simulated real-time environment. A system such as TIELT can minimize these irregularities by providing a more efficient implementation or by using low-pause garbage collection techniques.

Parallelism: TIELT is currently designed for evaluation of a single player learning agent, against an external or a human opponent, and there is no directly supported way to combine evaluations of several individuals in parallel into a single learning system, even though such evaluation is possible in the environment (Unreal Tournament supports up to 16 simultaneous players). Adding explicit multi-agent functionality to TIELT would greatly increase the utility of the platform when evaluating population-based learning systems like NEAT.

Support for Batch Experiments: While TIELT does provide some support for running experiments in batch mode, some settings are only available through the graphical user interface. This interactive component of TIELT makes it difficult to run long series of repeated experiments, especially when distributing the work to a cluster of machines. For some of our experiments, additional software was used to script user interactions in a virtual environment, which created unnecessary overhead. In designing a testbed such as TIELT, care should be taken to ensure that all use cases can be recreated in non-interactive batch mode.

TIELT has the capability to integrate the game engine, the TIELT application itself, and the decision system while they are running on different physical machines and communicate via network messages. This is a powerful feature, and it should be expanded with the ability to script experiments and to distribute evaluations over several different computers running TIELT. This would help optimize use of computational resources and researcher time.

Usability and Flexibility: In order to make TIELT integration less time-consuming, the framework can be simplified by making use of existing technology. Instead of using a custom scripting language, future integration systems can be made more powerful and easier to approach by using an existing scripting language such as Python or Ruby, bringing to bear existing documentation, libraries, and experience.

If the goal of the middleware is to support many different kinds of learning systems, its architecture should be flexible enough to be usable in all those models. The knowledge bases and modules of TIELT, while well-suited for rule-based learning, are not as useful with neuroevolution or online reinforcement learning. For NEAT as well as other reinforcement learning methods, the concepts of an evaluation episode, an individual agent, and a population are beneficial. Future versions of TIELT and other integration and evaluation systems can benefit from a more modular architecture which supplies some specific features needed by different kinds of learning agents and environments.

Access to Source: Learning agent benchmarking interfaces such as TIELT must have their source open to the users. Doing so will greatly shorten the debugging cycle as well as allow researchers to have complete knowledge of the implementation of their experimental system. Unfortunately, TIELT is currently a closed source project.

6 Conclusion

The results in this paper show that a generic framework such as TIELT can be used to integrate and evaluate adaptive decision systems with rich computer game environments. Basic target-seeking behavior was evolved with NEAT neuroevolution method for an agent in the Unreal Tournament video game. However, in order to make such frameworks practical and their use more widespread, progress needs to be made in several aspects. They must be designed and implemented as high-performance and lightweight applications, better utilize standard interfaces and existing scripting languages, and provide support for distributed and scripted operation for batch computational experiments. With these extensions, it may be possible to use sophisticated game playing domains in developing better exploration-based learning methods, as well as develop more interesting adoptive elements for future games.

A Acknowledgments

Thanks to David Aha and Matthew Molineaux for providing the TIELT platform. This work was supported in part by DARPA through an NRL grant #N00173041G025.

References

Agogino, A., Stanley, K., and Miikkulainen, R. (2000). Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38.

Aha, D. W., and Molineaux, M. (2004). Integrating learning in interactive gaming simulators. In *Challenges of Game AI: Proceedings of the AAAI'04 Workshop*. AAAI Press.

ESA (2005). Essential facts about the computer and video game industry.

Gold, A. (2005). Academic AI and video games: a case study... In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*. IEEE.

Gomez, F. (2003). *Robust Non-Linear Control Through Neuroevolution*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin.

Kaminka, G. A., Veloso, M. M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A. N., Scholer, A., and Tejada, S. (2002). Gamebots: a flexible test bed for multiagent team research. *Communications of the ACM*, 45(1):43–45.

Keogh, E., and Kassetty, S. (2002). On the need for time series data mining benchmarks: a survey and empirical demonstration. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 102–111.

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 and the 12th Annual Conference on Innovative Applications of Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Molineaux, M. (2004). *TIELT (v0.5 Alpha) User's Manual*.

Molineaux, M., and Aha, D. W. (2005). TIELT project website. Project web site, Naval Research Laboratory, `http://nrlsat.ittid.com/`.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005a). Evolving neural network agents in the NERO video game. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games(CIG'05)*. IEEE.

Stanley, K. O., Cornelius, R., Miikkulainen, R., D'Silva, T., and Gold, A. (2005b). Real-time learning in the NERO video game. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005) Demo Papers*.

Stanley, K. O., and Miikkulainen, R. (2002a). Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*. Piscataway, NJ: IEEE. In press.

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