Driving-Bots with a Neuroevolved Brain: Screaming Racers

Francisco Gallego, Faraón Llorens, Mar Pujol, Ramón Rizo

Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante
Ctra. San Vicente del Raspeig s/n, 03690 (Alicante)
{fgallego, faraon, mar, rizo}@dccia.ua.es

Abstract

The Videogame Industry of today is now just as strong as, and generates similar revenues to the Film Industry. Computer Games are distributed throughout the world and are sold to millions of people. Of the many different types of games, one of the most popular genres is car racing. Developers of this type of game are increasingly improving their Artificial Intelligence Systems so that their virtual drivers can exhibit Human-Level behaviours, and even higher. In this paper we show how these virtual drivers can be generically evolved using Neuroevolution, so obtaining several, distinct driving-bots with an increasing level of performance. These driving-bots can then be used as virtual opponents for different racing games, saving time and money in the development of a realistic AI System. To train our driving-bots and to test their performance we have developed Screaming Racers, a simple car-racing online videogame.

Keywords: Computer Games, Human-Level AI, Neuroevolution, Soft Computing.

1. Introduction

Videogames have evolved exponentially over the past twenty-five years. They have changed from simple bricks composed of flashing, single colour pixels with simple "left-right" computer controlled movements into extremely-detailed 3D models with thousands of polygons, bringing alive complicated bots with adaptive personalities, capable of making their own decisions and even of developing strategies.

Nowadays, our perception of how computer games are influencing our lives has changed completely. At present, games are able to bring to life entire virtual worlds with complex physics and social rules where thousands of human players, from every corner of the planet, and bots meet together every day [Bryce&Rutter03].

Within this scenario, lots of game genres have been developed since the arrival of the first video games. One of the most popular types is the Racing Genre, which has grown remarkably over recent years. In this class of game the player usually adopts the position of driver and is responsible for driving a car as fast as possible around a track, whilst competing against other human opponents and bots. In such a scenario, the challenge of creating realistic driving-bots in order to meet player's expectations and to offer a realistic challenge to the human is an interesting research field. Exploring this field we are the right way to find interesting Soft Computing algorithms, in an attempt to reproduce Human-Level behaviours and even improve them [van Lent et al. 99], [Laird&van Lent00], [Laird02].

In this paper we show how it is possible to evolve driving-bots to such a point that they exhibit Human-Level behaviours. In section 2 we present t he AI techniques most commonly used today for driving-bots and then introduce Neuroevolution and compare and contrast the different techniques. In section 3 we present Screaming Racers, the prototype videogame that has been developed to host our experiments and we discuss its internal design. Later, in section 4, we show how driving-bots can be Neuroevolved using Screaming Racers. Finally, sections 5 and 6 illustrate the results of our work, the conclusions we have made and ideas which will act as guidelines for future investigatory work.

2. Creating the AI Knight Rider

When we read about Artificial Intelligence Techniques for creating driving-bots, we find lots of approximations to the problem, but all of them try to solve it the straight way. It is usual to have a race-track divided in different sectors (Like Slot pieces), and structured in memory as a double linked list of sectors [Biasillo02a].

So, the easiest way to have an AI controlled car driving around a track is to define lines from the start to the end of every sector to guide the car along. By doing this, we can mark the optimal path on each track and the AI will simply follow the lines. We can also add extra information to each sector such as type of terrain or brake/throttle hint values.

But proceeding in this manner will only lead us to obtain a rigid, non-realistic driving-bot. Therefore, a second kind of approximation would be to use the optimal line merely as an aid, not to be followed exactly. Instead, it could have a set of rules designed to attempt to follow the line with some kind of error added to them.

In this world, driving-bots need to keep track of their location on the circuit, and to do this they use a finite-state machine (FSM) [Biasillo02b]. For example, a driving-bot may be in a state STATE_OFF_TRACK, which will encourage it to decide to return to the track, and will generate a path line from the present location point to a point in the middle of the optimal driving line of the sector where the car left the track.
These techniques are mixed with others that aim to tweak the parameters of optimal throttle, steering or braking in every sector and for every car type, thus avoiding the need for manual tweaking [Biasillo02c]. So, having a road map, an obstacle map and a list of cars it is possible to create apparently intelligent bots [Adzima02].

But all these techniques only succeed on the basis that the bots are supplied with information such as predefined driving-lines or other similar tricks, so producing the illusion of intelligence, but a million miles from real human-like intelligence. This is popularly known as cheating, and it has one major drawback: that the resultant AI system will be coupled to the game's logical system, because it would not be able to perform its tasks without adding the correct cheating data to every sector of every track.

2.1. A better approach

As our basic objective is to create generic bots with driving capabilities, this means they will need to be able to understand the surrounding environment and to make intelligent decisions based on this understanding. Moreover, our approach must avoid cheating, in order to decouple resulting bots from the game, letting them be used in other games and/or applications.

Therefore, we will provide each of our artificial driving-bots with a set of sensors to simulate the senses of a real human driver. Then, the data stemming from these sensors will be processed as input by an Artificial Neural Network (ANN) aimed to take decisions over the physics of the car (throttle, brake, steering, etc.). So, we will have a population of ANNs, each of them corresponding to a singular driving-bot brain. At this stage we will be able to evolve the entire population of brains, thus creating progressively better adapted brains. This process is known as Neuroevolution (NE) [Mandischer02], [Beyer&Schwefel02].

2.2. Neuroevolution to the rescue

NE methods focus their work on evolving neural networks through Genetic Algorithms. This is accomplished by encoding structural information and/or weights into a genome, and then evolving populations of genomes using a Genetic Algorithm (GA). So, by doing this, we obtain new different ANNs over time, from which we hope to find some that display characteristics that would make them useful for our purposes. In order to direct this evolutionary process in the right way, it is usual to adopt a reinforcement learning approach [Moriarty et al. 99].

The real problems here are twofold: firstly, how to design a good punishments/rewards system that encourages and takes advantage of reinforcement

learning and, secondly how to encode genomes in a suitable form for our GA.
We will concentrate initially on the more difficult problem, that of genome encoding.

2.2.1. Genome encoding schemes

NE researchers have developed lots of different approximations in their attempts to obtain the most favourable forms of encoding and evolving genomes for ANNs. The results of their work enables us to encode an ANN in different ways, such as a sequence of bit-strings (GENITOR) [Whitley89], as a binary matrix [Miller et al. 89], or as a generating grammar [Hussain&Browse98], for instance.

But all of these representations have one major drawback that cannot be overcome. This problem occurs when different genome representations produce ANNs that exhibit the same behaviour. In this case, a conventional crossover operator (i.e. a multipoint one) will produce poor quality offspring. This problem is known as the Competing Conventions Problem (CCP) [Hancock92].

Different attempts were made to solve the CCP (even avoiding the use of the crossover operator), but none of them provided sufficiently acceptable results until the advent of Neuroevolution of Augmenting Topologies (NEAT) [Buckland02], [Stanley&Miikkulainen02a, 02b], which was the first algorithm to use a global innovation database to track all the population genes, thus avoiding the confusion that gave rise to the CCP.

2.2.2. NEAT in a flash

NEAT uses two kinds of genes to describe the genome of an ANN: link genes and neuron genes. As may be guessed from their names, neuron genes describe different kinds of neurons and link genes describe the connections between two neuron genes.

NEAT makes use of continuous complexification, starting with the topologically-minimal genomes and adding neurons and links by means of its mutations operators. Hence, whenever a new structure is added to a genome, it is called an innovation. These innovations are stored in a global database. When a new neuron or link is being added to a genome, the database is referenced and an innovation ID is assigned to any new structures, so telling us if it was previously discovered by another genome, or if it is completely new.

At this stage, each of our genes among the population is globally identified by its innovation ID. So, to enable crossover, we can track genes of each parent genome chronologically by its innovation ID, matching same genes (the ones with the same innovation ID) on both parents before crossing them (Figure 1). Once aligned, NEAT does this: it iterates down the length of each genome of every parent, inheriting matching genes randomly from one of them, and also inheriting non-matching genes only from the fittest parent.

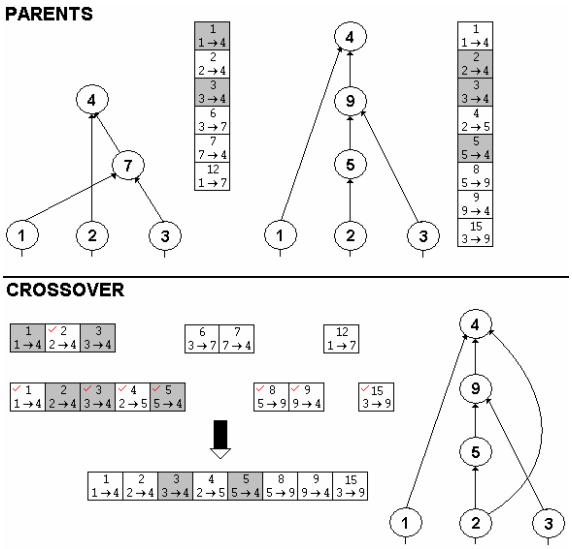

Figure 1. An example of NEAT's crossover (grey genes are disabled)

Moreover, NEAT's AG makes use of speciation, which has the advantage of protecting new innovations by isolating them reproductively from the rest of the population.

All of these characteristics make NEAT a powerful algorithm to explore the universe of behaviours with the aim of finding a desired one. In our tests, NEAT returned the best performance of the Neuroevolution algorithms we selected to compare.

3. Screaming Racers

Screaming Racers (SR) is a car racing simulation videogame where cars are controlled by artificially-intelligent driving-bots, which try to learn from their experiences and so improve their skills as pilots. The aim of the game is to create the best group of artificial driving-bots, which will become our personally managed motor-racing team. In order to accomplish this task, we will have to train our bots using the tools provided by the game. Once we have created and trained our personal motor-racing team, we can test their effectiveness and efficiency by taking part in tournaments against other teams.

In order to train our driving-bots, Screaming Racers has a set of possibilities we can explore:

- We can create our custom-defined set of race-tracks. To define a new track we have to create a new text file with the track definition in our Track and Car's Description Language (LDC, "*Lenguaje de descripción de coches y circuitos*"). In this language, a basic circuit is simply composed by its name, its friction rate and a list of consecutive lines which define track sectors.
- We can also create a custom-defined set of cars using LDC. A basic car is defined by its name, a set of physics parameters (acceleration, speed, spin, and friction rate), a front color and a set of polygons to define its appearance.
- From the definitions of race-tracks and cars we can construct training plans for bots as lists of activities. One activity is characterized by one track where the activity will take place and a set of control parameters (num. of laps to clear, time to finish, type of car to use, and num. of reiterations).
- We can tweak all the parameters concerning the evolution of our pilot's brain (mutation rate, crossover rate, maximum number of neurons, weight ranges, reinforcement punishments and rewards, and so on).

By playing around and adjusting all of these possibilities we can construct specially-designed training plans with the intention of maximizing the learning rate of our driving-bot population. Once our driving-bots are trained, we can save their brains to a file, which allows us to continue training them at a later date, or group them together as a team and challenge other teams of driving-bots or human players.

3.1. Internal design

Screaming Racers has been designed as a multi-agent online videogame where players use a client application to connect to a server which realizes simulations of the races. This multi-agent design of the driving-bots population provides several advantages:

- Driving-bots can migrate from client to server and vice versa whenever needed, thus minimizing the problems related to latency upon communications.
- Driving-bots can sense the environment using a set of defined sensors, which decouples them from the internal representation of this environment.
- Driving-bots can communicate between themselves, letting teams share knowledge in order to benefit from a collaborative approach.

In our client/server architecture (Figure 2), the main components of Screaming Racers are those inside the Server Kernel. These components are responsible for creating and maintaining the virtual environment and the bots' multi-agent system, as well as communicating with clients. The description of the tasks given to every component is as follows:

- Physics engine: Its main purpose is to apply physics laws of the virtual environment to all the objects in the world. These physics laws do not have to be identical to the real laws of physics, and may be tuned for convenience (i.e. they can be simplified to realize experiments).
- Neuroevolution engine: This component implements different Neuroevolution algorithms used to train and evolve the population of driving-bots.
- Simulator Agent: This agent is responsible for creating and maintaining the virtual environment needed for every race or training session. It manages the center of simulation where the action occurs, so it has to deal with the race-track, the cars, the time control, the statistics of the race, the objects of the environment, etc.
- Graphics engine: Is responsible for showing the action graphically on the screen. It also manages the graphical interface to communicate with the user or the server administrator, whatever the case may be.
- Client/Server Manager: Oversees the remaining system components and acts as the boss. It coordinates the labours of the other components and tells them what they have to do, also acting as scheduler (i.e. Tells the Simulator when to start running and which environment characteristics to simulate)

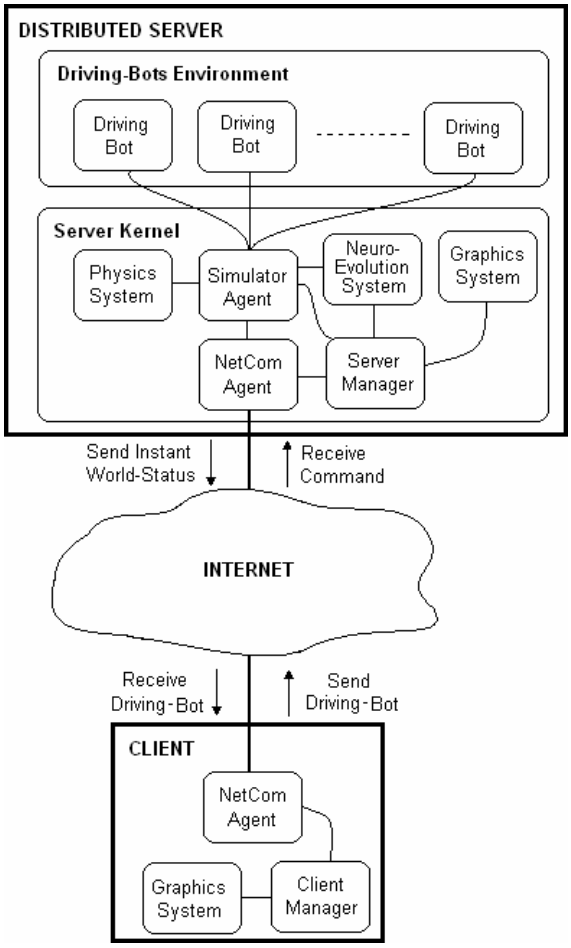

Figure 2. Screaming Racers' internal design

- NetCom Agent: It performs all the tasks needed for the server to communicate with the clients and vice versa, whenever required. This includes receiving and sending pilot agents, sending present-instant world-status, etc.
- Driver-bot: Each instance of this class of agent is the implementation of the brain of one of the bots running in the simulation. So, these agents receive from the environment (the Simulator Agent) the information coming from their sensors and have to reply with the desired actuator changes, i.e. accelerating by an amount of "x" m/s^2

4. Neuroevolving driving-bots with SR

As we have stated in previous sections of this paper, in order to train our bots, we previously need to define several factors:

- A sensor / actuator model.
- A punishment / reward system.
- ANNs genome encoding.
- AG's operators (mutation, crossover)

For the ANNs genome-encoding and AG's operators, we use those defined by every algorithm we apply. So, in order to understand how NE increases driver-bots' intelligence, let's look at how both the sensor model and the punishments/rewards system are defined in the Screaming Racers' environment

4.1. Sensor/actuator Model

Once we had implemented SR, we wanted to compare the effectiveness of the different Neuroevolution algorithms in order to know which one performs the task of learning to drive a race car best So, the first thing we did was to select a common sensor model for the driving-bots, to be used it in all the different tests. The selected sensor model is that illustrated in Figure 3.

In this sensor model, the lines which emerge from the sides of the car represent the lines of vision that a real driver would have if he were driving the car. The three front lines represent the lines of vision of the driver looking through the windscreen, whereas the four rear lines indicate the driver's view using his mirrors. The two lines projecting from the side of the car represent his view looking through the side windows. In Figure 3, the white lines are those lines of vision that have detected the boundary of the race track, while the black lines represent the lines of vision that perceive only clear road.

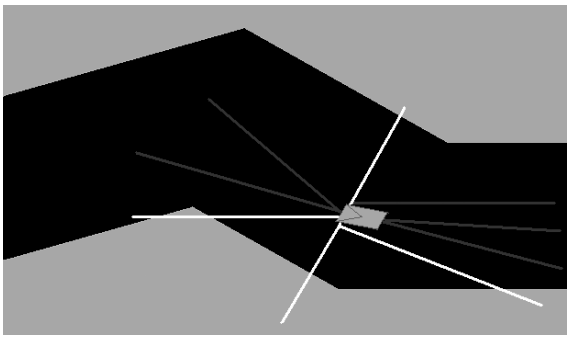

Figure 3. Selected sensor model

When a sensor detects a race-track boundary or an obstacle, it gives a [0, 1) normalized value to the driving-bot's brain, which tells them the distance from the car to the obstacle (a 1 would represent a sensor detecting clear road). With this information, bots can "see" their environment and so consequently make choices and decisions. These choices are simply decided by the two actuators every driving-bot has. These actuators are the two

output neurons of every brain ANN, which represent the [0, 1] normalized throttle and steering amounts.

4.2. Punishments/rewards system

The punishments/rewards system we have created aims to give credit to several actions considered laudable in a driver, and to prevent our bots from repeating behaviours thought to be harmful. So, the actions/behaviours we consider are (C = Gives credit, P = Receives punishment):

- Going straight, following race-track (C).
- Maintaining a high level of speed (C).
- Passing from one sector to the next (C).
- Reducing speed below an established level (P).
- Continuously over-steering (P).
- Crashing into boundaries, objects or cars (P).

All of these rewards/punishments are controlled by a set of control parameters which can be tweaked by the Screaming Racer player. It is possible to activate and deactivate the effects of these events as well as giving more importance to one or more of them, which permits the exploration of different possibilities.

5. Results

In this section we show a comparison of the results obtained by applying several distinct NE algorithms to the problem of evolving our driving-bots. We have carried out the same tests on each of the algorithms we have selected, and the results clearly favour Full NEAT above all others.

The tests carried out consisted in evolving driving-bots along 500 generations on the same race-track (each one during 75 seconds) using a population of 100 brains and the same control parameters (mutation and crossover rate, punishment/rewards coefficients, etc). Results of these tests are summarized in Figure 4. These results refer to the evolution of the fitness of the best brain inside the population in every algorithm. To give an indication of what this fitness value means, a human with a good level is able to obtain a score of approx. 1800 fitness points while the best human-performers have results of around 2250 fitness points. Full NEAT (NEAT including speciation) achieves a maximum of 1925.19 fitness points, which is considerably higher than a "good" human and is approaching the levels of the very best.

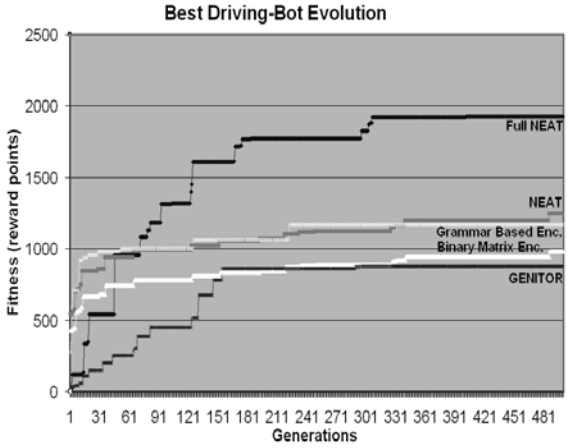

Figure 4. Results obtained with selected methods

To graphically show the skill level acquired by the best driving-bot, Figure 5 indicates a trace image of the bot taking one of the curves of the race-track without crashing into the boundaries, and with a genuine control of steering, throttle and braking.

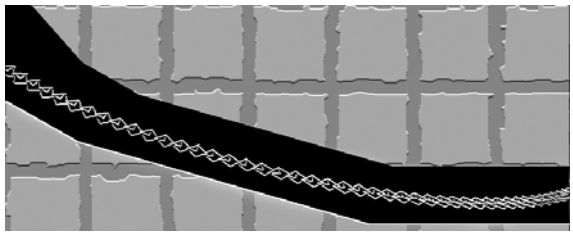

Figure 5. A curve taken by best driving-bot.

6. Conclusions and further work

In this paper we have talked about computer games and, more specifically, about Artificial Intelligence Systems for driving-bots in Racing Videogames. We have exposed the usual methods of achieving some kind of realistic AI for these environments and we have then shown how Neuroevolution can be an improved way of obtaining similar and even better results. We also have explained our recently-developed videogame-prototype called Screaming Racers, which has provided us the means to generically train driving-bots using different NE approaches and compare results.

The final results of the experiments show us how NE and NEAT can be useful in bringing driving-bots to life, with the best bots obtained even able to outperform normal level human-players. Moreover, if the process is repeated continuously and the brains from the best individual of every generation are saved, we could use them as different opponents with increasing levels of difficulty inside a racing game. Furthermore, as these bots have been generically trained, they could show their abilities in any other game, with the only restriction being that the game provided the bot with a similar set of sensors to "understand" the environment.

In future research, we will add complex physics, more complex environments and more realistic cars. We also plan to create virtual racing robots which emulate real robots in order to use Screaming Racers as a research environment to conduct racing experiments with emulated robots. We also intend to improve NEAT and attempt to outperform the best human players with our driving-bots and their Neuroevolved brains.

Acknowledgments

We would like to thank specifically the work done by Hector Linares, who has collaborated in the implementation of the Graphics engine of the Screaming Racers prototype.

References

[Adzima02] Adzima, J. C. (2002): Competitive AI Racing under Open Street Conditions. AI Game Programming Wisdom. Charles River Media. Pages 460-471.

[Biasillo02a] Biasillo, G. (2002): Representing a Racetrack for the AI. AI Game Programming Wisdom. Charles River Media. Pages 439-443

[Biasillo02b] Biasillo, G. (2002): Racing AI Logic. AI Game Programming Wisdom. Charles River Media. Pages 444-454.

[Biasillo02c] Biasillo, G. (2002): Training an AI to Race. AI Game Programming Wisdom. Charles River Media. Pages 455-459.

[Bryce&Rutter03] Bryce, J. & Rutter, J. (2003) The Gendering of Computer Gaming: Experience and Space, in Fleming, s. & Jones, I. Leisure Culture: Investigations in Sport, Media and Technology. Leisure Studies Association, pp. 3-22.

[Buckland02] Buckland, Mat. (2002): AI Techniques for game programming. Game Development Series. Premier Press.

[Laird02] Laird, John E. (2002): Research in human-level AI using computer games. Commun. ACM 45: 32-35.

[Laird&van Lent00] Laird, John E.; van Lent, Michael (2000): Interactive Computer Games: Human-level AI's Killer Application. National Conference on Artificial Intelligence (AAAI), Austin, Texas.

[Beyer&Schwefel02] Beyer, H.G. and Schwefel, H.P. (2002): Evolution strategies: Acomprehensive introduction. Natural Computing 1(1), pp. 3–52.

[Hancock92] Hancock, P. J. B. (1992): Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. Proc. Int. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92), D. Whitley and J. D. Schaffer, Eds., Los Alamitos, CA: IEEE Comput. Soc. Press, 1992, pp. 108—1.

[Hussain&Browse98] Hussain, T. S., Browse, R. A. (1998): Network generating attribute grammar encoding. IEEE International Joint Conference on Neural Networks. Anchorage, Alaska.

[Mandischer02] Mandischer, M. (2002): A comparison of evolution strategies and backpropagation for neural network training. Neurocomputing 42(1–4), pp. 87–117.

[Miller et al. 89] Miller, G.F., Todd, P.M., and Hedge, S.U. (1989): Designing Neural Networks Using Genetic Algorithms. In J.D. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pages 379-384. Morgan Kaufmann.

[Moriarty et al. 99] Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999): Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research 11, pp. 199–229.

[Stanley&Miikkulainen02a] Stanley, Kenneth O., Miikkulainen, Risto (2002): Evolving neural networks through augmenting topologies. Evolutionary Computation 10, pages 99-127

[Stanley&Miikkulainen02b] Stanley, Kenneth O., Miikkulainen, Risto (2002): Efficient Evolution Of Neural Network Topologies, Proceedings of the Congress on Evolutionary Computation (CEC '02). Piscataway, NJ: IEEE, 2002.

[van Lent et al. 99] van Lent, M.; Laird, J. E.; Buckman, J.; Hartford, J; Houchard, S.; Steinkraus, K.; and Tedrake, R. (1999): Intelligent Agents in Computer Games. Proceedings of the National Conference on Artificial Intelligence, Orlando, FL, pp. 929-930.

[Whitley89] Whitley, D. (1989): The Genitor Algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. Proceedings of the Third International Conference on Genetic Algorithms and their Applications. Massachusetts Institute of Technology, 116-121.