The 2009 Simulated Car Racing Championship

Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lönneker, Luigi Cardamone, Diego Perez, Yago Sáez, Mike Preuss, and Jan Quadflieg

***Abstract*—In this paper, we overview the 2009 Simulated Car Racing Championship—an event comprising three competitions held in association with the 2009 IEEE Congress on Evolutionary Computation (CEC), the 2009 ACM Genetic and Evolutionary Computation Conference (GECCO), and the 2009 IEEE Symposium on Computational Intelligence and Games (CIG). First, we describe the competition regulations and the software framework. Then, the five best teams describe the methods of computational intelligence they used to develop their drivers and the lessons they learned from the participation in the championship. The organizers provide short summaries of the other competitors. Finally, we summarize the championship results, followed by a discussion about what the organizers learned about 1) the development of high-performing car racing controllers and 2) the organization of scientific competitions.**

***Index Terms*—Car racing, competitions.**

Manuscript received December 11, 2009; revised February 25, 2010; accepted April 26, 2010. Date of publication May 18, 2010; date of current version June 16, 2010. This work was supported in part by the IEEE Computational Intelligence Society (IEEE CIS) and the ACM Special Interest Group on Genetic and Evolutionary Computation (ACM SIGEVO). The simulated car racing competition of the 2009 ACM Genetic and Evolutionary Computation Conference (GECCO) was supported by E. Orlotti and NVIDIA. The work of D. Perez and Y. Saez was supported in part by the Spanish MCyT project MSTAR, Ref:TIN2008-06491-C04-03. The work of D. Pelta was supported by the Spanish Ministry of Science and Innovation under Project TIN2008-01948 and the Andalusian Government under Project P07-TIC- 02970.

D. Loiacono, P. L. Lanzi, and L. Cardamone are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan 20133, Italy (e-mail: loiacono@elet.polimi.it; lanzi@elet.polimi.it; cardamone@elet.polimi.it).

J. Togelius is with the IT University of Copenhagen, 2300 Copenhagen S, Denmark (e-mail: julian@togelius.com).

E. Onieva is with the Industrial Computer Science Department, Centro de Automática y Robótica (UPM-CSIC), Arganda del Rey, 28500 Madrid, Spain (e-mail: enrique.onieva@car.upm-csic.es).

D. A. Pelta is with the Computer Science and Artificial Intelligence Department, Universidad de Granada, 18071 Granada, Spain (e-mail: dpelta@decsai.ugr.es).

M. V. Butz and T. D. Lönneker are with the Department of Cognitive Psychology, University of Würzburg, Würzburg 97070, Germany (e-mail: butz@psychohologie.uni-wuerzburg.de; thies.loenneker@stud-mail.uni-wuerzburg.de).

D. Perez was with the University Carlos III of Madrid, Leganes, CP 28911 Madrid, Spain. He is now with the National Digital Research Center, The Digital Hub, Dublin 8, Ireland (e-mail: diego.perez.liebana@gmail.com).

Y. Sáez is with the University Carlos III of Madrid, Leganes, CP 28911 Madrid, Spain (e-mail: yago.saez@uc3m.es).

M. Preuss and J. Quadflieg are with the Chair of Algorithm Engineering, Computational Intelligence Group, Department of Computer Science, Technische Universität Dortmund, Dortmund 44227, Germany (e-mail: mike.preuss@cs.uni-dortmund.de; jan.quadflieg@cs.uni-dortmund.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2010.2050590

I. Introduction

During the last three years, several simulated car racing competitions have been organized in conjunction with leading international conferences. Researchers from around the world submitted car controllers for a racing game; the controllers were evaluated by racing against each other on a set of unknown tracks; the one achieving the best results won. In 2009, the *first* simulated car racing championship was organized as a joined event of three conferences: 2009 IEEE Congress on Evolutionary Computation (CEC, Trondheim, Norway), the 2009 ACM Genetic and Evolutionary Computation Conference (GECCO, Montréal, QC, Canada), and the 2009 IEEE Symposium on Computational Intelligence and Games (CIG, Milan, Italy). The championship consisted of nine races on nine different tracks divided into three legs, one for each conference, involving three Grand Prix competitions each. Competitors were allowed to update their drivers during the championship by submitting a different driver to each leg. Each Grand Prix competition consisted of two stages: the qualifying stage and the main race. In the qualifying stage, each driver raced alone for a fixed amount of time. The eight drivers that performed best during the qualifying stage moved to the main race event, which consisted of five laps. At the end of each race event, drivers were scored using the Formula 1 (F1)[1] point system. Winners were awarded based on their scores in each conference competition. At the end, the team that scored the most points over all three legs won the championship.

In this paper, we overview the 2009 Simulated Car Racing Championship. In Section II, we describe the background of the competition and previous competitions related to it. Then, we describe the software framework developed for the competition in Section III, while in Section IV, we describe the competition rules. In Section V, the authors of the five best controllers describe their own work while the organizers briefly describe the other competitors. In Section VI, we report the results of the competition, and in Section VII, we discuss what we learned from the competitions regarding both 1) the design of competitive car racing controllers and 2) the organization of a scientific competition involving artificial intelligence in a game context. Finally, in Section VIII, we discuss the future of the competition.

II. Background

A. Previous Work on Simulated Car Racing

This series of competitions did not arise out of a vacuum; for the last six years, a substantial number of papers have been published about applying computational intelligence techniques to

[1]http://www.formula1.com/

simulated car racing in one form or another. Many, but not all, of these papers are due to the organizers and participants in this competition. In what is probably the first paper on the topic, neural networks were evolved to drive a single car on a single track as fast as possible in a simple homemade racing game [1]. The paper also established the approach of using simulated rangefinder sensors as primary inputs to the controller, which has been adopted in most subsequent papers and in this competition. A number of papers using versions of the same experimental setup investigated, for example, the learning of racing skills on multiple tracks, competitive coevolution of controllers, imitation of human driving styles, and evolution of new racing tracks to suit human driving styles; summaries of these developments are available in [2] and [3]. Later papers have investigated different approaches to imitating human driving styles [4], [5], generating interesting driving [6], and online learning [7].

B. Previous Game-Related Competitions

The simulated car racing competitions are part of a larger group of competitions that have been organized over the last few years in association with the IEEE CEC and the IEEE CIG in particular. These competitions are based on popular board games (such as *Go* and *Othello*) or video games (such as *Pac-Man*, *Super Mario Bros*, and *Unreal Tournament*). In most of these competitions, competitors submit controllers in some programming language, which interfaces to an application program interface (API) built by the organizers of the competition. The winner of the competition usually is the person or the team that submitted the controller that played the game best, either on its own (for single-player games such as *Pac-Man*) or against others (in adversarial games such as *Go*). Usually, prizes of a few hundred U.S. dollars are associated with each competition and a certificate is always awarded. Usually, there is no requirement that the submitted controllers exploit methods of computational intelligence algorithms but in many cases the winners turn out to include such computational intelligence methods in some form or another. Some of these competitions have been very popular with both conference attendants and the media, including coverage in mainstream news channels such as *New Scientist* and *Le Monde*. The submitting teams tend to comprise students, faculty members, and persons not currently in academia (e.g., working as software developers).

A number of guidelines for how to hold successful competitions of these sorts have gradually emerged from the experience of running these competitions. These include having a simple interface that anyone can get started within a few minutes' time, being platform and programming language independent whenever possible, and open-sourcing both competition software and submitted controllers.

There are several reasons for holding such competitions as part of the regular events organized by the computational intelligence community. A main motivation is to improve benchmarking of learning algorithms. Benchmarking is frequently done using very simple testbed problems, which may capture some aspects of the complexity of real-world problems. When researchers report results on more complex problems, the technical complexities of accessing, running, and interfacing to the benchmarking software might prevent independent validation of and comparison with the published results. Here, competitions have the role of providing software, interfaces, and scoring procedures to fairly and independently evaluate competing algorithms and development methodologies.

Another strong incentive for running these competitions is the stimulation of particular research directions. Existing algorithms get applied to new areas and the effort needed to participate in a competition is (or at least should be) less than it takes to come up with the results for a new problem, writing a completely new paper. Competitions might even bring new researchers into the computational intelligence fields, both academics and nonacademics. Another admittedly big reason for the stimulating effect, especially for game-related competitions, is that it simply looks cool and often produces admirable videos.

In 2007, simulated car racing competitions were organized as part of the IEEE CEC and the IEEE CIG. These competitions used a graphically and mechanically simpler game. Partly because of the simplicity of the software, these competitions enjoyed a good degree of participation. The organization, submitted entries, and results of these competitions were subsequently published in [8].

In 2008, simulated car racing competitions were again held as part of the same two conferences, as well as of the ACM GECCO conference. Those competitions were similar to those of the year before in their overall idea and execution, but there were several important differences. The main difference was that the event was built around a much more complex car racing game, the open-source racing game *The Open Racing Car Simulator (TORCS)*. While the main reason for using this game was that the more complex car simulations (especially the possibility for having many cars on the track at the same time with believable collision handling) poses new challenges for the controllers to overcome, other reasons included the possibility of convincing, e.g., the game industry that computational intelligence algorithms can handle "real" games and not only academically conceived benchmarks and the increased attention that a more sophisticated graphical depiction of the competition generates (see Fig. 1).

The 2009 championship was technically very similar to the 2008 competitions, using the same game and a slightly updated version of the competition software package. Our efforts went into simplifying the installation and usage of the software, sorting out bugs and clarifying rules rather than adding new features. The real evolution has been in the submitted controllers, the best of which have improved considerably and, as can be seen from the descriptions below, now constitute state-of-the-art applications of computational intelligence (CI) techniques for delivering high-performing solutions to a practical problem.

III. The Competition Software

In this section, we briefly describe the competition software we developed for the championship as an extension of the *TORCS* game, which we first adopted for the 2008 World Congress on Computational Intelligence (WCCI) and the 2008 CIG simulated car racing competitions. In particular, we overview *TORCS* and illustrate the modifications we did to introduce an

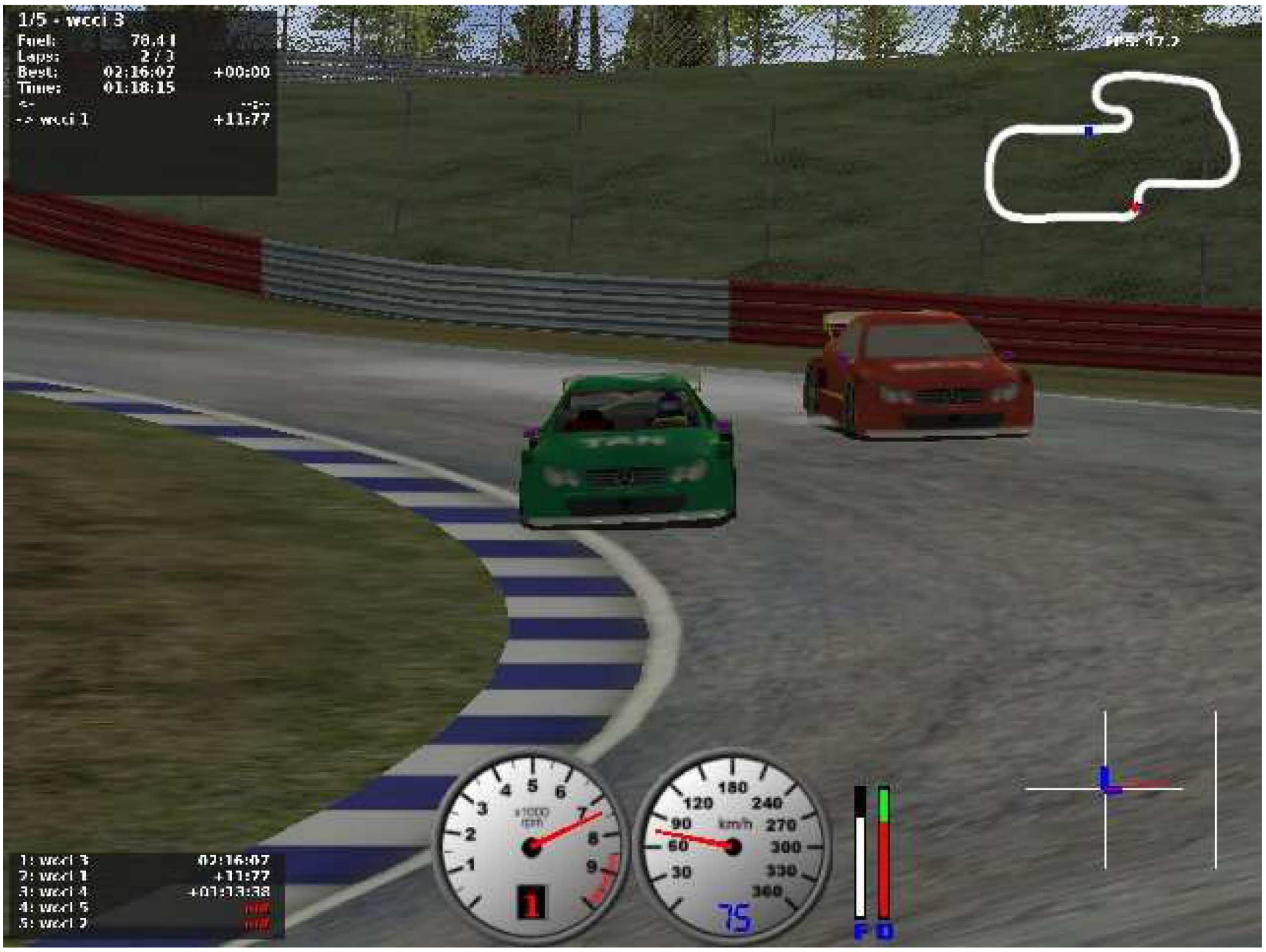


Fig. 1. Screenshot from a *TORCS* race.

unified sensor model, real-time interactions, and independence from a specific programming language.

A. The TORCS Game

TORCS [9] is a state-of-the-art open-source car racing simulator. It falls somewhere between being an advanced simulator, like recent commercial car racing games, and a fully customizable environment, like the ones typically used by researchers in computational intelligence for benchmarking purposes. On the one hand, TORCS is the best free alternative to commercial racing games in that:

i) it features a sophisticated physics engine, which takes into account many aspects of the racing car (e.g., collisions, traction, aerodynamics, fuel consumption, etc.);
ii) it provides a rather sophisticated 3-D graphics engine for the visualization (Fig. 1);
iii) it also provides a lot of game content (i.e., several tracks, car models, controllers, etc.), resulting in a countless number of possible game situations.

On the other hand, *TORCS* has been specifically devised to allow the users to develop their own car controllers, their own *bots*, as separate C++ modules, which can be easily compiled and added to the game. At each control step (game tick), a bot can access the current game state, which includes information about the car and the track as well as the other cars on the track; a bot can control the car using the gas/brake pedals, the gear stick, and the steering wheel. The game distribution includes many programmed bots, which can be easily customized or extended to build new bots. *TORCS* users developed several bots, which often compete in international competitions such as the driver championship[2] or those organized by the *TORCS* racing board.[3]

B. Extending TORCS for the Championship

TORCS comes as a standalone application in which the bots are C++ programs, compiled as separate modules, which are loaded into main memory when a race takes place. This structure has three major drawbacks with respect to the organization of a scientific competition. First, races are not in real time since bots' execution is blocking: if a bot takes a long time to decide what to do it simply blocks the game execution. This was an issue also afflicting the software used in earlier car racing competitions (e.g., the one organized at the 2007 IEEE CEC). Second, since there is no separation between the bots and the

[2]http://speedy.chonchon.free.fr/tdc/

[3]http://www.berniw.org/trb/

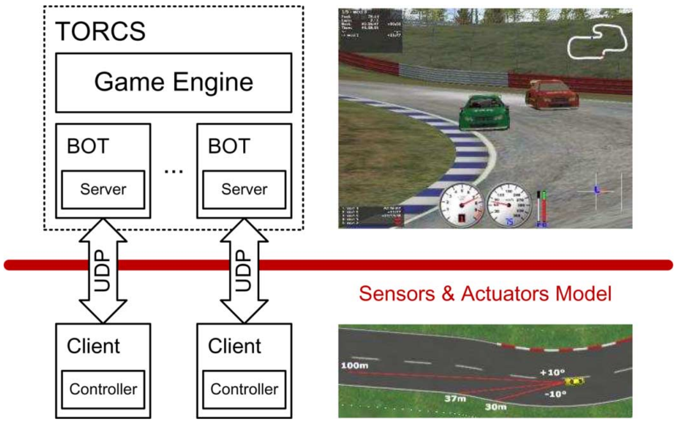
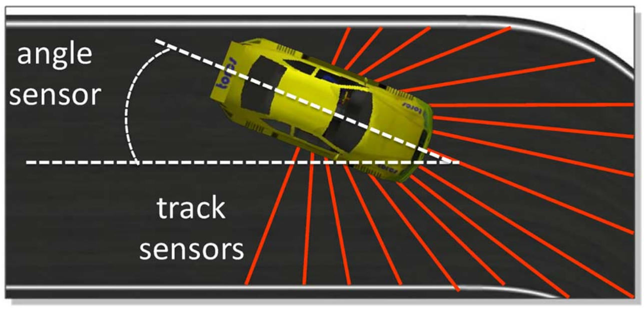


Fig. 2. Architecture of the API developed for the competition.

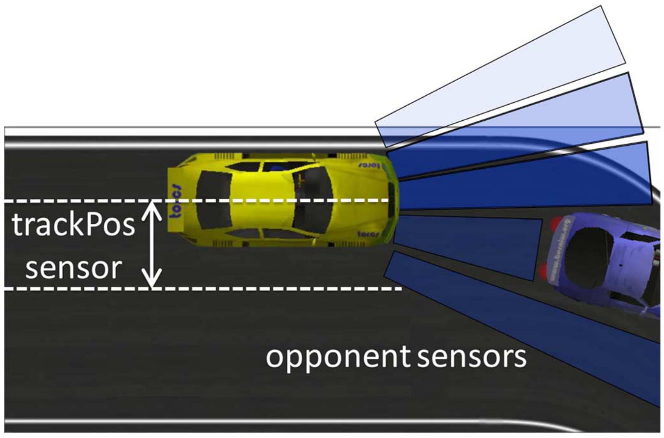
simulation engine, the bots have full access to all the data structures defining the track and the current status of the race. As a consequence, each bot can use different information for its driving strategy. Furthermore, bots can analyze the complete state of the race (e.g., the track structure, the opponents position, speed, etc.) to plan their actions. Accordingly, a fair comparison among methods of computational intelligence, using the original *TORCS* platform, would be difficult since different methods might access different pieces of information. Last but not least, *TORCS* restricts the choice of the programming language to C/C++ since the bots must be compiled as loadable modules of the main *TORCS* application, which is written in C++.

The software for the 2009 Simulated Car Racing Championship extends the original *TORCS* architecture in three respects. First, it structures *TORCS* as a client–server applications: the bots run as external processes connected to the race server through UDP connections. Second, it adds real time: every game tick (which roughly corresponds to 20 ms of simulated time), the server sends the current sensory inputs to each bot and waits for 10 ms (of real time) to receive an action from the bot. If no action arrives, the simulation continues and the last performed action is used. Finally, the competition software creates a physical separation between the driver code and the race server, building an abstraction layer, that is, a sensors and actuators model, which 1) gives complete freedom of choice regarding the programming language used for bots and 2) restricts the access to the information provided by the designer.

The architecture of the competition software is shown in Fig. 2. The game engine is the same as the original *TORCS*; the main modification is the new server bot, which manages the connection between the game and a client bot using UDP. A race involves one server bot for each client; each server bot listens on a separate port of the race server. At the beginning, each client bot identifies itself with a corresponding server bot establishing a connection. As the race starts, each server bot sends the current sensory information to its client and awaits an action until 10 ms (of real time) have passed. Every game tick, which corresponds to 20 ms of simulated time, the server updates the state of the race. A client can also request special actions (e.g., a race restart) by sending a message to the server.



(a)



(b)

Fig. 3. Details of four sensors: (a) angle and track sensors; (b) trackPos and four of the 36 opponent sensors.

Each controller perceives the racing environment through a number of sensor readings, which reflect both the surrounding environment (the tracks and the opponents) and the current game state. A controller can invoke basic driving commands to control the car. Table I reports the list of available sensors; Table II reports all available control actions (see [10] for additional details); Fig. 3 shows the four sensors in detail (angle, track and trackPos and four of the 36 opponent sensors). Controllers had to act quickly on the basis of the most recent sensory information to properly control the car; a slow controller would be inherently penalized since it would be working on lagged information.

To further facilitate the participation in the competition, a client with simple APIs as well as a sample programmed controller were provided for C++ and Java languages and for Windows, Mac, and Linux operating systems.

IV. Rules and Regulations

The 2009 Simulated Car Racing Championship was a joined event comprising three competitions held at 1) the 2009 IEEE CEC, 2) the 2009 ACM GECCO, and 3) the 2009 IEEE CIG. The championship consisted of nine races on nine different tracks divided into three legs, one for each conference, involving three Grand Prix competitions each. Teams were allowed to update their driver during the championship by submitting a different driver to each leg. Each leg consisted of two stages: the qualifying and the actual Grand Prix races. During the qualifying stage, each driver raced alone for 10 000 game ticks, which corresponds to approximately 3 min and 20

TABLE I
DESCRIPTION OF AVAILABLE SENSORS

Name	Description
angle	Angle between the car direction and the direction of the track axis.
curLapTime	Time elapsed during current lap.
damage	Current damage of the car (the higher is the value the higher is the damage).
distFromStartLine	Distance of the car from the start line along the track line.
distRaced	Distance covered by the car from the beginning of the race
fuel	Current fuel level.
gear	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
lastLapTime	Time to complete the last lap
opponents	Vector of 36 sensors that detects the opponent distance in meters (range is [0,100]) within a specific 10 degrees sector: each sensor covers 10 degrees, from $-\pi$ to $+\pi$ around the car.
racePos	Position in the race with respect to other cars.
rpm	Number of rotation per minute of the car engine.
speedX	Speed of the car along the longitudinal axis of the car.
speedY	Speed of the car along the transverse axis of the car.
track	Vector of 19 range finder sensors: each sensors represents the distance between the track edge and the car. Sensors are oriented every 10 degrees from $-\pi/2$ and $+\pi/2$ in front of the car. Distance are in meters within a range of 100 meters. When the car is outside of the track (i.e., trackPos is less than -1 or greater than 1), these values are not reliable!
trackPos	Distance between the car and the track axis. The value is normalized w.r.t to the track width: it is 0 when car is on the axis, -1 when the car is on the left edge of the track and +1 when it is on the right edge of the car. Values greater than 1 or smaller than -1 means that the car is outside of the track.
wheelSpinVel	Vector of 4 sensors representing the rotation speed of the wheels.

TABLE II
DESCRIPTION OF AVAILABLE EFFECTORS

Name	Description
accel	Virtual gas pedal (0 means no gas, 1 full gas).
brake	Virtual brake pedal (0 means no brake, 1 full brake).
gear	Gear value.
steering	Steering value: -1 and +1 means respectively full left and right, that corresponds to an angle of 0.785398 rad.
meta	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

s of the actual game time. The eight drivers that covered the largest distances qualified for the actual Grand Prix races. The actual races took place on the same three tracks used during the qualifying stage. The goal of each race was to complete five laps finishing in first place. At the end of each race, the drivers were scored using the F1 system: ten points to the first controller that completed the five laps, eight points to the second one, six to the third one, five to the fourth one, four to the fifth one, three to the sixth one, two to the seventh one, and one to the eighth one. In addition, the driver performing the fastest lap in the race and the driver completing the race with the smallest amount of damage received two additional points each. Winners were awarded based on their scoring in each conference competition. At the end, the winner of the 2009 Simulated Car Racing Championship was the team who scored the most points summed over all three legs.

V. THE COMPETITORS

Thirteen teams participated in the championship. Five teams updated their drivers between competitions; all the other seven teams submitted one driver; two of these teams participated only in the last leg held at the 2009 IEEE CIG. In this section, the best five teams describe their controllers at length highlighting: 1) what methods of computational intelligence they used for online and offline training, 2) how the development of the controller was structured, 3) the main challenges they faced, 4) their main successes of the approach they followed, and 5) the main strength and weaknesses of their controller with respect to the other competitors. At the end of this section, we also give brief descriptions of the other seven competitors.

A. Enrique Onieva and David A. Pelta

The idea behind this bot is to have a driving architecture based on a set of simple controllers. Each controller is implemented as an independent module in charge of a basic driving action; each module is rather simple and intuitive so that it is very easy to modify and tune. In particular, the architecture consists of six modules for 1) gear control, 2) target speed, 3) speed control, 4) steering control, 5) learning, and 6) opponents management. *Gear control* shifts between gears and also interacts with a "car stuck" detector, by applying the reverse gear. *Target speed* determines the maximum allowed speed on a track segment. *Speed control* uses the throttle and brake pedals to achieve the speed determined by the *target speed* module. It also implements a traction control system (TCS) and an antilock brake system (ABS) to avoid slipping, by modifying actions over the throttle and the brake. *Steering control* acts over the steering wheel. The *learning module* detects the segments of the circuit where the target speed can be increased or reduced; these are typically long straight segments or segments near bends with a small curvature radius. *Opponents management* applies changes over the steering, gas, and brake outputs to adapt the driving actions when opponents are close.

1) Development: The first architecture submitted was improved after the 2009 IEEE CEC competition [11] with the addition of the learning and the opponents management modules.

The learning module became necessary after observing that the car systematically went off track in the same points every lap. In particular, the TCS was added because when the car went off of the track axis, it occasionally slipped and lost control (possibly getting stuck). We also made several simple modifications to the other modules to improve the performance. The overall improvement due to these modifications was impressive as the recovery from an off-track position turned out to be much costlier than the slowing down to keep the car on track. After few initial tests, we also realized the importance of keeping an adequate speed in each track segment, which allows the bot to drive as fast as possible while staying within the track limits. Instead of defining *a priori* target speed values, we computed these values using a fuzzy-logic-based Mamdani controller [12]. This module is implemented as a fuzzy-rule-based system using a computational model of the ORdenador Borroso EXperimental (Fuzzy Experimental Computer; ORBEX) fuzzy coprocessor [13]. ORBEX implements fuzzy systems with trapezoidal shapes for input variables and singleton shapes for output variables. These models provide a simplification in the operations necessary to carry out the fuzzy inference [14], which makes them a good choice to deal with the low response times required by autonomous driving tasks [15]–[17]. A simple handmade fuzzy controller with seven rules was implemented. Results showed a good performance as well as a very simple and interpretable structure. The results for racing without opponents were very encouraging when compared with those obtained by the controllers of the competitions held at the 2008 IEEE CEC and the 2008 IEEE CIG. The performance in many of the tracks provided by the *TORCS* engine was also evaluated, to assess the correct behavior in critical curve combinations. We also tried to improve the parameters of the fuzzy system using a genetic algorithm [18]–[20] but at the end no clear benefits were achieved.

Also the *opponents management* module turned out to be critical, as racing involves many known and unknown factors. For example, as the strategy of an opponent is unknown, the decision for braking or overtaking might be a matter of "good luck." In the first races involving opponents, the bots provided in the *TORCS* distribution were very fast in comparison to our controller; accordingly, our controller was usually overtaken early, and then it would race alone for the remainder of the race. Later, to study different racing situations, we conducted an in-depth analysis of the behavior of our bot when up to 20 opponents were present. In all the experiments, our bot would always start in the last position of the grid and we monitored the position achieved at the end of the race, the damage suffered, and the time per lap for all the racers. As a result, we implemented a basic racing strategy involving behaviors to overtake opponents, avoid collisions by a sudden steering movement, and speed reduction before an imminent collision. At the end, it was observed that just one or two of the opponents (implemented by *TORCS*' bots) were able to achieve a performance comparable to our architecture.

2) Strengths and Weaknesses: The main strength of our proposal lies in its simplicity and in its highly parametric design, which allows for further improvements by tuning procedures based on soft computing methods. In addition, the architecture of the opponents management module makes it possible to perform multiple overtakes while suffering little damage and allows for the addition of new behaviors (e.g., a behavior to obstruct being overtaken). Finally, the introduction of a basic learning module reduces the chances to repeat previous mistakes so as to significantly improve the lap time during subsequent laps. As for the weaknesses, our bot currently does not have a global "race strategy" to, for example, be cautious in the initial laps and aggressive in later ones, or to increase the target speed when it is in the bottom positions during the race. In our opinion, the design of such a strategy can lead to a faster and more efficient driver.

B. Martin V. Butz and Thies D. Lönneker—COBOSTAR

The COgnitive BOdySpaces for *TORCS*-based Adaptive Racing (COBOSTAR) racer combines the idea of intelligent sensory-to-motor couplings [21] with the principle of anticipatory behavior [22]. It translates maximally predictive sensory information into the current target speed and the desired driving direction. The differences between current and target speeds and desired and current driving directions then determine the appropriate control commands.

Adhering to the principle of anticipatory behavior and downscaling the sensory space, the basic behavioral control on track considers only the distance and the angle of the longest distance-to-track sensor (cf., the "track" sensor in Table I) that is pointing forward. Off track, where this information is not available, control is based on the angle-to-track axis information and the distance from the track (cf., "angle" and "trackPos" sensors in Table I). Both mappings were optimized by means of computational intelligence techniques. Moreover, various other features were added before and after the optimization process, including the antislip regulation (ASR), the ABS, the stuck monitor with backup control, the jump controller, the crash detector for online learning, and the adaptive opponent avoidance mechanism.

1) Computational Intelligence for Offline Learning: The mapping was optimized by means of the covariance matrix adaptation (CMA) evolution strategy [23]. A complex function mapped the used sensory information onto the target angle and the target speed [24]. The fitness of a controller was the distance raced when executing 10 000 game ticks. The on- and off-track mapping functions were optimized in separate evolutionary runs.

Various evaluations showed that the resulting optimized parameter values were not globally optimal and sometimes not even optimal for the optimized track. Thus, it was necessary to do a final evaluation stage in which the most general parameter set was determined. We chose those parameter values as the final control values that yielded the longest covered distance averaged over all available tracks in the *TORCS* simulator.

2) Computational Intelligence for Online Learning: Besides the offline optimization of the basic sensory-to-motor mapping, we also developed several online adaptation techniques to improve the driving behavior while driving multiple laps with opponents.

Seeing the available information, it is generally possible to adjust behavior in the second lap based on the experience gathered in the first lap. In fact, theoretically, it is possible to scan the track in the first lap and thus reach a near-optimal behavior in the second lap. Our approach, however, was slightly more cognitively inspired with the aim of adapting a behavior in subsequent laps given a severe crash in a previous lap. Dependent on the severity of the crash and the controller behavior in the steps before the crash, the target speed in subsequent laps was lowered before the crash point. Adjustment parameters were hand-tuned in this case.

Moreover, we employed an opponent monitor mechanism in order to keep track of the relative velocities of the surrounding cars. To do so, the distance sensors to opponents had to be tracked over time and had to be assigned to fictional car objects. The resulting relative velocity information was then essential to employ effective opponent avoidance. Treating opponents essentially as moving obstacles, we projected the estimated time until a crash occurs onto the distance-to-track sensors, if this time was smaller than the time until a crash into the corresponding track edge. Using these adjusted distance-to-track sensor values, the steering angle and the target speed were determined as usual. The result was an emergent opponent avoidance behavior, which proved to be relatively robust and effective.

3) Development: The general controller development started with the design of the sensory-to-motor mapping functions. Next, CMA optimization was completed for the on-track mapping using a rudimentary off-track control strategy. With the evolved parameters, we then optimized the off-track mapping and finally did another round of on-track optimization. At this last stage, we also cross evaluated the evolved parameter sets on other tracks and chose the most generally successful setting for the submissions to the competition. While we left the general sensory-to-motor mappings untouched over the three competitions in 2009, we improved various other behavioral aspects.

4) Challenges and Successes: After COBOSTAR's success at the 2009 IEEE CEC, we mainly worked on further parameter optimization, which however did not yield any real performance improvements. Meanwhile, we overlooked the importance of opponent avoidance, especially because the racers available in the *TORCS* simulator itself all drive rather well so that serious crashes due to opponent interactions were very rare. The third place at the 2009 ACM GECCO competition convinced us that opponent avoidance must be the crucial factor (especially also seeing that COBOSTAR was first in the qualifying stage) for success in the final competition during the 2009 IEEE CIG. The idea of an opponent monitor and the subsequent projection of the opponents onto the distance-to-track sensors yielded a solution that required the least other modifications of the controller. Additionally, though, we added or improved several other strategy aspects including jump detection, stuck recovery, off-track speed reduction, and crash adaptation in subsequent laps. The success at the 2009 IEEE CIG proved that the modifications were worthwhile.

5) Strengths and Weaknesses: The strength of our controller lies in its simplicity and the use of the most informative sensory information. For control, anticipatory information is clearly most effective—and in the *TORCS* simulation, this is the distance-to-track sensory information. Moreover, the indirect mapping from sensors to target speeds and then to the actual throttle or break application proved effective, yielding smooth and effective vehicle behaviors. While a strength might also be seen only in taking the maximum distance sensor information into account, a weakness certainly is that no additional information was considered. For example, the distance sensors next to the farthest may contain additional information about the exact radius of the curve ahead. Our biggest competitor [11] did use this additional information, which may be the reason why we were partially beaten by their controller. Nonetheless, also their controller used indirect distance-to-track-based sensory-to-motor mappings and thus generally the same control principle. The robustness additions we added to our controller and especially the adaptive opponent avoidance made our controller marginally superior in the final leg of the competition.

C. Luigi Cardamone

This controller is a slightly modified version of the winner of the 2008 IEEE CIG simulated car racing competition [25], [26]. The idea behind our approach is to develop a competitive driver from scratch using as little domain knowledge as possible. Our architecture consists of an evolved neural network implementing a basic driving behavior, coupled with scripted behaviors for the start, the crash–recovery, the gear change, and the overtaking.

In neuroevolution, the choice of the network inputs and outputs plays a key role. In our case, we selected the inputs which provide information directly correlated to driving actions, that is, the speed and the rangefinder inputs. We selected two outputs: one to control the steering wheel and one to control both the accelerator and the brake. When the car is on a straight stretch, the accelerator/brake output is ignored and the car is forced to accelerate as much as possible. This design forces the controller to drive as fast as possible right from the beginning and prevents the evolutionary search from wasting time on safe but slow controllers.

Opponent management, including overtaking, is a crucial feature of a competitive driver. In our case, we decided to adopt a hand-coded policy which adjusts the network outputs when opponents are close. On a straight stretch, our policy always tries to overtake. When facing a bend, our policy always brakes to avoid collisions if opponents are too close. This simple policy gave very good results during the race. In fact, our driver performed better than faster controllers with less reliable overtaking policies.

Gear shifting and crash recovery are also managed by two scripted policies, borrowed from bots available in the *TORCS* distribution.

Our first controller was not equipped with a policy for the race start. In the first leg, at the 2009 IEEE CEC, we realized that the start was crucial in that 1) several crashes usually occur, which can severely damage cars, and 2) several overtakes are possible, which can lead to a dramatic improvement of the final result. Accordingly, since the 2009 ACM GECCO, we introduced a very simple hand-coded strategy for the race start that basically tries to overtake all the other cars, as soon as possible by moving on one side of the track.

1) Computational Intelligence for Offline Learning: To evolve the neural network for our controller, we applied neuroevolution of augmenting topologies (NEAT) [27], one of the most successful and widely applied methods of neuroevolution. NEAT [27] works as the typical population-based selecto-recombinative evolutionary algorithm: first, the fitness of the individuals in the population is evaluated; then selection, recombination, and mutation operators are applied; this cycle is repeated until a termination condition is met. NEAT starts from the simplest topology possible (i.e., a network with all the inputs connected to all the outputs) and evolves both the network weights and structure.

In our approach, each network is evaluated by testing it for one lap. The fitness is a function of the distance raced, the average speed, and the number of game ticks the car spent outside the track [25], [26]. The learning was performed just on one track, namely, Wheel 1 [9], which presents most of the interesting features available in the tracks distributed with *TORCS*.

2) Strengths and Weaknesses: In our opinion, the major strength of our approach is that it required just a little domain knowledge to produce a competitive driver. In fact, even the hand-coded policies we used can be evolved from scratch with very little effort (see, for instance, [28]). Our approach also seems to provide a high degree of generalization in that it performed reasonably well even if training was carried out on one track only.

The main weakness of our approach is that it tends to evolve too careful controllers, which drive at the center of the track most of the time. Accordingly, our approach cannot produce drivers following the optimal track trajectory. Another weakness is the lack of any form of online adaptation: once the best controller is deployed, no further learning takes place. However, we are currently working toward adding some sort of online learning at different stages [7].

D. Diego Perez and Yago Saez

The main idea behind our controller is to design an autonomous vehicle guidance system that can tackle a wide variety of situations. To deal with such a difficult goal, the driver's logic has been organized in three modules: 1) a finite state machine (FSM), 2) a fuzzy logic module, and 3) a classifier module.

The *FSM* module is intended to provide the controller with a sense of state; thus, once the controller has been appropriately trained, it can *remember* whether it is preparing for tasks such as taking a turn, overtaking another car, or veering off the track. This approach allows the designers to code appropriate behaviors inside the states.

The *fuzzy logic* module retrieves information from the sensors and utilizes it to move between the states of the FSM. Fuzzy logic systems are well-known and widely used methods for building controllers [29]. The one developed here works as the one presented in [30].

The *classifier* module selects a subset of the input sensors and tries to predict the type of track stretch the car is in, such as a bend, a straight, or the approach to or departure from a turn. The predicted class is used to guide state transitions in the FSM.

1) Computational Intelligence for Offline Learning: Two methods of computational intelligence were applied offline to build the controller. The *J48* decision tree builder [31] was used for the classifier module using as inputs the angle between the car and the track and the distances to the edges. The decision tree outputs a class value which indicates whether the car is on a straight stretch, on a bend, or close to a bend. In addition, the shapes of the fuzzy sets and some internal parameters of the FSM were tuned offline using a multiobjective genetic algorithm (more precisely, nondominated sorting genetic algorithm II (NSGA-II) [32]). The evolutionary algorithm was applied to minimize both the lap time and also the number of mistakes made while driving (e.g., the damage suffered, the time spent off track, etc.).

2) Computational Intelligence for Online Learning: The proposed architecture is not designed for online learning and thus no method of computational intelligence was applied while driving.

3) Development: The fundamental assumption underlying the development of this controller is that if the driver knows 1) what type of stretch is facing and 2) what the car is doing (e.g., taking a turn, recovering from an off-track position, etc.), then the driver can take the most appropriate driving decisions. Accordingly, one of the first tasks we addressed was the identification of the track shape. For this purpose, we gathered a large amount of training data by capturing all the input sensors using several tracks for which the different stretches were manually identified. We tested several sets of attributes and several classifiers (e.g., PART, *J48*, neural networks, K-means, etc. [33]); at the end, we selected a decision tree built by *J48*, which achieved an overall accuracy of the 97% using just the data regarding the car's angle and distance to track edges. Then, we built a set of fuzzy rules to interpret the other sensory data. Fuzzy sets were used to determine several race situations such as *being outside, running very fast, being oriented to the left*, or *having an edge very close*. The shape of this set was initially defined by hand and eventually tuned using an evolutionary algorithm.

When all the input data have been transformed into fuzzy sets and classes, we developed an FSM with four states: *run, prepare turn, turn*, and *backtotrack*. The transitions among these states are triggered by the discretized inputs obtained in the previous stage. In addition, the current state is updated during each game tick and generates the values to update the actuators (i.e., throttle and steering). Finally, the offline training process is performed by applying NSGA-II to optimize the system parameters so as to minimize both the lap time and the number of mistakes. For this purpose, we used four tracks with a wide variety of characteristics. At the end of the optimization process, as all the controllers from the Pareto front were optimal, we had them competing and selected the best one to participate in the actual competition.

4) Challenges and Successes: A major challenge we faced during the development of our controller was the generation of the data set to build the classifier. For this purpose, we created a basic controller to collect racing data affected by as little noise as possible. Since the controller would collect a huge amount of information, the attribute selection process was crucial to this phase. Notwithstanding the several challenges, at the end, we obtained a classifier with an impressive 97% accuracy, which we consider a great success. As the development also included a reliability system to ignore possible noise, the resulting classes were accurately obtained for almost all classifications.

5) Strengths and Weaknesses: The main strength of our approach is that the controller, at each cycle, has information about its state (i.e., about what it is doing) and where it is. This sort of self-consciousness allows it to recover easily from an incident, to come back on the track if the car is outside, or to prepare for a turn that is coming ahead.

The main weakness is that it was initially designed for autonomous vehicle guidance. Consequently, it performs quite well with safe driving on almost all the tracks we tested. However, it is rather careful for a race car competition which usually needs to brake or accelerate fully. Due to the fuzzy system developed here, it is almost impossible to find full-braking or full-acceleration outputs. This is an important problem when racing against opponents. To deal with this issue, we plan to design specific sets of rules to modify the fuzzy system in the next versions of our driver.

E. Jan Quadflieg and Mike Preuss—Mr. Racer

This controller comprises different modules tackling specific driving aspects. Some of the modules implement simple hand-coded heuristics, e.g., the module activating the recovering behavior, the steering module that keeps the car on the track, and the opponent avoidance module. The controller simply focuses on driving as fast as possible and does not pay attention to the opponents, as long as they are not too close. The main idea underlying the development of this controller is that to be competitive a driver must be able 1) to detect the type of track stretch that the car is immediately heading to and then 2) to approach the forthcoming track segment with an appropriate speed. We implemented these two features as independent components: one *bend-detection* heuristic which classifies track segments and a *speed matching* component learned offline. The interface between the two components has been deliberately chosen to be human interpretable and consists of six types of track segments: straight, straight approaching turn, full speed turn, medium speed turn, slow turn, and hairpin turn.

The *bend-detection* classifier works as follows. At first, the 19 forward track sensors are converted to vectors that have the car as origin. Then, the longest vector on the right-hand side and the longest vector on the left-hand side are selected by scanning from the outermost vector inward. When the car is on a straight stretch that is not too wide, the same vector is selected twice. When approaching a turn, the vectors are different. The angles of all the vectors on the right-hand side and the left-hand side up to the selected longest vectors are added up. The result is a value which we map onto the discrete track segment classes using a hand-coded rule. The steering heuristic has been borrowed from Kinnaird–Heether (described in [34]) and slightly modified by tripling the steering angle for in hairpin bends while doubling it in slow bends.

1) Computational Intelligence for Offline Learning: Once the class of the approaching track segment is reliably detected by the *bend-detection* classifier, it has to be matched with an appropriate speed. We applied a simple $(1+1)$ evolution strategy for adapting a speed table on a given track. The table consists of six rows for the segment types and five columns of speed values spread over the range of feasible values, namely, 34, 102, 170, 238, and 306 km/h. Each entry contains a real variable bound between -1 meaning full brake and 1 standing for full acceleration. For driving according to the table, we look up the two entries corresponding to the two speed values best matching our current speed and interpolate the reaction linearly. Starting from a rather conservative setting with a full acceleration only set for low speeds, the table is evolved by driving a fixed time (around one to three laps, depending on the speed) for each newly generated individual and measuring the distance covered.

While experimenting, we noted that the most robust controllers are evolved when a track with a wide variety of features (e.g., possibly including all different track segment types) is chosen for learning.

2) Computational Intelligence for Online Learning: Online learning is not incorporated into this controller yet. This is deemed as a future project when the potentials of offline learning are fully explored.

3) Development: The modular design of our controller stems from the two principle assumptions that 1) to simplify development, not too many different tasks should be treated at once; and 2) APIs should be human interpretable to enable programmers to check whether a specific behavior is coherent and comparable to the actions of a human driver.

4) Challenges and Successes: We faced two major challenges which we successfully solved 1) to come up with a good heuristic to recognize the type of the approaching track segment and 2) to define a good encoding for the offline learning of the speed control.

5) Strengths and Weaknesses: Mr. Racer is very good at detecting the different types of bends and at staying on track. However, it still drives too slowly from time to time and it is not fully capable of dealing with opponents.

F. The Other Competitors

In addition to the five best competitors, seven additional teams entered the championship at various stages.

1) Chung-Cheng Chiu, Academia Sinica, Taipei, Taiwan, submitted a hand-coded and hand-tuned controller. The steering works by minimizing the angle between the direction of the car and the direction of the track sensor which returns the largest distance to the edge. Thus, the steering moves the car toward the wider surrounding empty space. The speed control and the stuck detection are adapted from the ones of the example controller included in the competition software package.
2) Jorge Munoz, Carlos III University of Madrid, Madrid, Spain, submitted a hand-coded controller with 29 parameters which were tuned by an evolutionary algorithm. Each set of parameters was evaluated by using it to drive on ten different tracks for 10 000 game ticks each. The fitness was computed as a function of the distance raced and the top speed and more precisely as (distRaced/10 000) + (topSpeed/400). The controller is an updated version of the one described in [35].
3) Dana Vrajitoru and Charles Guse, Indiana University South Bend, South Bend, submitted a hand-coded controller. The desired speed is computed with a basic approach that scales the speed based on the *distance to the next bend*. The desired steering is computed based on the *type of the next bend*: a hand-tuned heuristic is applied to identify sharp bends using the track sensors.
4) Paolo Bernardi, Davide Ciambelli, Paolo Fiocchetti, Andrea Manfucci, and Simone Pizzo, University of Perugia,

TABLE III
RESULTS OF THE SECOND EVALUATION STAGE OF THE 2009 IEEE CEC LEG; SCORES ARE COMPUTED AS THE MEDIAN OVER SIX RACES

Competitor	Michigan Speedway		Alpine2		Corkscrew		Total
	Distance	Score	Distance	Score	Distance	Score	Score
Butz & Lönneker	13516.9	10	7798.0	10	7147.8	10	30
Onieva & Pelta	11797.7	6	6826.4	8	3377.7	5	19
Quadflieg & Preuss	9926.6	3	6398.6	6	6900.7	8	17
Chiu	11987.4	8	5527.7	3	3695.0	6	17
Cardamone	11301.3	5	6389.5	5	2569.5	3	13
Perez & Saez	10946.5	4	6272.0	4	3235.2	4	12
SimpleDriver	6233.4	-	3216.9	-	4142.3	-	-
Berniw	12956.0	-	7432.4	-	8256.4	-	-

Perugia, Italy, submitted a reactive rule-based controller constructed partly through the imitation of human driving styles. An array of sensors including speed, track angle, and edge distances was discretized, leading to a total of 685 900 possible sensor states. Logs of human gameplay were then used to infer rules from each sensor state to the appropriate actions. This rule set was subsequently manually tuned.

5) Ka Chun Wong, the Chinese University of Hong Kong, Hong Kong, submitted a hand-coded controller, called *simplicity*, that implements a relatively straightforward mechanism to compute the desired speed and the desired direction using both track edge and opponent sensors. The controller also deals with special cases (e.g., when the car is outside the asphalt or it is stuck) and includes an ABS filter for braking. Interestingly, the controller memorizes the points along the track where crashes occurred by storing a list of the *distFromStartLine* values for each crash; in subsequent laps, the controller slows down when approaching one of such points.
6) Witold Szymaniak, Poznań University of Technology, Poznań, Poland, submitted a reactive controller represented as a directed acyclic graph, which was evolved with Cartesian genetic programming [36] implemented using the evolutionary computation in Java (ECJ) package [37]. The set of inputs is restricted to angle to track, front track sensor, speed, lateral speed, and *distRaced*) and two composite inputs relating to track curvature; most rangefinder sensors are never used directly. The function nodes implement standard arithmetic, trigonometric, and conditional functions. Outputs are interpreted as the desired speed and angle. The gear shifting was taken from the example Java controller and was coupled with a custom crash recovery mechanism and ABS. The fitness used during evolution was rather complicated and took into account the distance raced (relative to an example reference controller), the damage taken, and the difficulty of the track segment.
7) Marc Ebner and Thorsten Tiede, Eberhard Karls Universität of Tübingen, Tübingen, Germany, submitted a controller evolved with genetic programming using the ECJ package [37]. The controller consists of two evolved trees: one controlling the steering angle and one controlling acceleration and deceleration. The controller inputs were defined as a subset of the sensors provided by the competition software. The fitness function was computed as the average performance on a set of five different tracks. The detailed description of the controller is provided in [38].
8) Wolf-Dieter Beelitz, BHR Engineering, Pforzheim, Germany, submitted a rather sophisticated hand-coded controller. This was developed by adapting one of the best controllers, Simplix,[4] available in the *TORCS* community [9] to the setup of the Simulated Car Racing Competition.

VI. RESULTS OF THE COMPETITION

The championship was organized in three *legs*, each one held at a major conference: the 2009 IEEE CEC, the 2009 ACM GECCO, and the 2009 IEEE CIG. Each leg involved three Grand Prix competitions on three previously unknown tracks; each Grand Prix competition consisted of two stages: the qualifying stage and the actual race.

In the first qualifying stage, each controller raced alone in each of the three tracks and its performance was measured as the distance covered in 10 000 game ticks (approximately, 3 min and 20 s of the actual game time). For each track, the controllers were ranked according to the distance covered (the higher the covered distance, the better the rank) and then scored using the F1 point system.[5]

The eight controllers which received the highest total score during the qualifying stage moved to the next stage and raced together in each one of the three tracks. Each race consisted of five laps. Controllers were scored using the F1 point system based on their arrival order. In addition, two bonus points were awarded both 1) to the controller that achieved the fastest lap during the race and 2) to the controller that suffered the least amount of damage during the race. To obtain a reliable evaluation, for each track, we performed eight races using eight different starting grids. The first starting grid was based on the scores achieved by the controllers during the qualifying. In the next seven races, the starting grid was generated by *shifting* the previous grid as follows: the drivers in the first seven positions were moved backward by one position, while the driver in the

[4]http://www.wdbee.gotdns.org:8086/SIMPLIX/SimplixDefault.aspx

[5]http://www.formula1.com/

TABLE IV
RESULTS OF THE QUALIFYING STAGE AT THE 2009 IEEE CEC LEG; STATISTICS ARE MEDIANS OVER TEN RUNS

Competitor	Michigan Speedway	Alpine2	Corkscrew	Total
Butz & Lönneker	12	7.5	9	28.5
Onieva & Pelta	8	9	5	22
Cardamone	5	10	5	20
Quadflieg & Preuss	3	6	10	19
Perez & Saez	6	4	6	16
Chiu	5	5	4	14

last position was moved to the first position. As a result, every driver had the chance to start the race in every position of the grid. For each track, the final score of a driver was computed as the median of its scores over the eight races performed on the same track. Note that only this score was considered for the purpose of the championship.

A. *Results of the First Leg (The 2009 IEEE CEC)*

The first leg of the championship was part of the competition program of the 2009 IEEE CEC, chaired by Simon Lucas. It involved two tracks taken from the *TORCS* distribution (Michigan Speedway and Alpine 2) and one track (Corkscrew) available from the Speed Dreams website,[6] a fork of the *TORCS* project. We received five submissions respectively from 1) Enrique Onieva and David A. Pelta (Section V-A), 2) Martin V. Butz and Thies D. Lönneker (Section V-B), 3) Diego Perez and Yago Saez (Section V-D), 4) Jan Quadflieg and Mike Preuss (Section V-E), and 5) Chung-Cheng Chiu (Section V-F). To compare the new submissions with the previous editions, we also included the champion of the previous competition by Luigi Cardamone (Section V-C) to the pool.

Table IV reports the results of the qualifying stage and includes (as a reference) the performance of the example driver (*SimpleDriver*) distributed with the competition software and the best hand-coded controller provided in *TORCS* (*Berniw*). All the submitted controllers outperformed the driver provided with the competition software. This result *per se* represents an improvement with respect to the previous competitions [34] where only few entries performed better than the simple controller. More importantly, for the first time, a competitor (i.e., the driver by Butz and Lönneker) was able to outperform one of the fastest bots with full information access (Berniw) in two tracks out of three. The same driver was also clearly the best performing among the six entries, whereas there was no clear difference among the other drivers. However, there was an overall improvement with respect to the previous edition in that the previous champion scored worse than all the competitors except for Perez and Saez (see Table IV).

[6]http://speed-dreams.sourceforge.net/

The second stage of the evaluation process was performed including all the submitted entries, as they are fewer than the eight available slots. Accordingly, for each track, we ran only six races instead of the eight planned. Table IV reports the scores of all the entries in each track. The winner of the qualifying stage is also the winner of this second stage: in fact, the entry of Butz and Lönneker achieved overall the highest score winning the first leg of the championship. However, as can be noted, the differences among the drivers are now dramatically reduced and, except for the first two positions, the results in Table IV do not confirm the results of the qualifying (Table IV).

B. *Results of the Second Leg (The 2009 ACM GECCO)*

The second leg was organized in conjunction with the 2009 ACM GECCO. It involved three tracks available in the *TORCS* distribution (Dirt 3, Alpine 2, and E-Road) and, for the first time, a rather challenging dirt track was included. We received five additional submissions (see Section V-F) from 1) Paolo Bernardi and colleagues, 2) Dana Vrajitoru and Charles Guse, 3) Ka Chun Wong, 4) Witold Szymaniak, and 5) Jorge Munoz. In addition, we received updates from four teams: 1) Butz and Lönneker, 2) Perez and Saez, 3) Cardamone, and 4) Onieva and Pelta.

Table V reports the results of the qualifying stage. When compared to the results of the first leg, the performance of the top controllers is now much closer, whereas the difference between the top controllers and the other ones is more evident. The top controllers are still very competitive with respect to Berniw, one of the best programmed controllers available in *TORCS*. In fact, the driver by Butz and Lönneker outperforms Berniw on the E-Road track and the driver by Onieva and Pelta has a very similar performance on the Dirt 3 track. Furthermore, all the submitted controllers except two outperformed the SimpleDriver (the only exceptions being the Chiu's and the Szymaniak's controllers in the Dirt 3 track). Unfortunately, during the qualifying stage, the driver by Quadflieg and Preuss was not able to complete the evaluation process in the E-Road track because the controller software crashed repeatedly during the process. Accordingly, this controller did not qualify for the second evaluation stage.

Table VI reports the outcome of the second stage. Surprisingly, the results did not confirm the outcome of the first stage completely in that the fastest controller (Butz and Lönneker) was outperformed by slower controllers. The analysis of the videos from the races suggests that slower controllers implemented a better opponent and crash management than COBOSTAR and they could deal more effectively with complex situations (for instance, during the race start). As a result, the second leg was won by Onieva and Pelta, who achieved the highest score over the three races.

C. *Results of the Third Leg (The 2009 IEEE CIG)*

The final leg was held during the 2009 IEEE CIG. It involved one track (Forza) from the *TORCS* distribution and two tracks

TABLE V
RESULTS OF THE QUALIFYING OF THE 2009 GECCO LEG; STATISTICS ARE MEDIANS OVER TEN RUNS

Competitor	Dirt 3		Alpine		E-Road		Total Score
	Distance	Score	Distance	Score	Distance	Score	
Butz & Lönneker	6216.2	8	8677.1	10	10253.3	10	28
Onieva & Pelta	6659.8	10	8386.5	8	9301.1	8	26
Cardamone	5439.8	4	7308.0	4	7716.3	5	13
Perez & Saez	5190.4	3	7661.0	6	6586.6	2	11
Bernardi et al.	4993.9	1	7184.2	3	8036.8	6	10
Vrajitoru & Guse	5788.3	6	6410.3	0	6384.4	1	7
Wong	5106.5	2	6470.0	1	6638.3	3	6
Munoz	4459.1	0	5548.7	0	7573.5	4	4
Chiu	901.6	0	6539.0	2	5124.3	0	2
Szymaniak	782.8	0	3711.9	0	4501.7	0	0
Quadflieg & Preuss	5502.0	5	7482.2	5	NQ	-	NQ
SimpleDriver	3267.9	-	3376.7	-	3999.7	-	-
Berniw	6808.2	-	9216.6	-	9670.5	-	-

TABLE VI
RESULTS OF THE SECOND STAGE OF THE 2009 GECCO LEG; SCORES ARE MEDIANS OVER EIGHT RACES

Competitor	Alpine	E-Road	Dirt 3	Total
Onieva & Pelta	10	10	12	32
Cardamone	7	8	8	23
Butz & Lönneker	5.5	7.5	3.5	16.5
Vrajitoru & Guse	5.5	3.5	4	13
Wong	3.5	4	5	12.5
Bernardi et al.	3.5	5	2.5	11
Perez & Saez	4	3.5	3.5	11
Munoz	3	2.5	3	8.5
Quadflieg & Preuss	0	0	0	0
Chiu	0	0	0	0
Szymaniak	0	0	0	0

(Buzzard Raceway and Migrants) from the Speed Dreams website.[7] We received two additional submissions from 1) Marc Ebner and Thorsten Thiede, and 2) Wolf-Dieter Beelitz. We also received updates from 1) Butz and Lönneker, 2) Perez and Saez, 3) Onieva and Pelta, 4) Munoz, and 5) Quadflieg and Preuss.

Table VII reports the results from the qualifying rounds. As can be noted, the results are similar to the ones of the second leg held at the 2009 GECCO (Table V). However, the gap between the best controllers and the other ones is now significantly reduced. Most importantly, all the drivers outperformed the SimpleDriver in all the tracks and the top controllers performed similarly or even better than Berniw (e.g., in the Migrants track). Table VI reports the results from the actual races. Interestingly, the results of the races are coherent with the results of the qualifying rounds. In particular, the driver by Butz and Lönneker and the driver by Onieva and Pelta confirmed to be the best controllers: the final leg was won by Butz and Lönneker, who scored just one point more than Onieva and Pelta.

[7]http://speed-dreams.sourceforge.net/

D. Final Standings

Table IX reports the scores of the three legs and the final score achieved during the three legs of the championship in the final standings of the championship. Enrique Onieva and David Pelta won the 2009 Simulated Car Racing Championship. COBOSTAR, by Butz and Lönneker, which won the first and third leg, was the runner up. The winner of the previous competition held at the 2008 IEEE CIG, by Luigi Cardamone, finished in third place, followed by all the other drivers that entered the championship from the beginning.

E. Discussion

The quality of the controllers submitted to the championship is encouraging. All the controllers performed significantly better than the participants of the previous editions. The best competitors performed similarly to Berniw, one of the best controllers distributed with *TORCS*. Interestingly, most of the competitors significantly improved their performance along the championship. In particular, the best performing drivers in the championship were also the ones that improved their opponents and crash management capabilities the most.

VII. LESSONS LEARNED

In this section, the organizers would like to share what they believe they have learned from the organization of the 2009 Simulated Car Racing Championship.

A. On the Development Process

Since the 2008 WCCI, the simulated car racing competition has been based on *TORCS* and used the same API so that all the controllers submitted so far could be compared directly. This allowed the competitors to evolve and tune their approaches over

TABLE VII
RESULTS OF THE QUALIFYING OF THE 2009 CIG LEG; STATISTICS ARE COMPUTED AS THE MEDIAN OVER TEN RUNS

Competitor	Forza		Buzzard Raceway		Migrants		Total Score
	Distance	Score	Distance	Score	Distance	Score	
Butz & Lönneker	11768.3	6	9459.6	10	9603.0	10	26
Onieva & Pelta	11997.2	8	9255.9	8	9140.9	8	24
Munoz	10577.6	5	8377.6	5	8734.9	6	16
Quadflieg & Preuss	12191.5	10	7700.6	4	7730.6	0	14
Beelitz	10409.2	3	8425.8	6	8515.5	5	14
Chiu	10424.9	4	7262.2	2	8157.4	4	10
Cardamone	9862.2	2	7360.9	3	7858.8	2	7
Perez & Saez	9028.3	0	6919.1	0	8018.5	3	3
Ebner & Thiede	9666.5	0	7182.0	1	7766.8	1	2
Wong	9781.6	1	6183.5	0	6852.9	0	1
Vrajitoru & Guse	7176.7	0	6245.8	0	6362.8	0	0
Szymaniak	6856.8	0	4560.3	0	4939.6	0	0
Bernardi et al.	6685.9	0	4688.7	0	5973.2	0	0
SimpleDriver	6122.7	-	4035.3	-	4396.0	-	-
Berniw	12939.9	-	10158.3	-	9502.3	-	-

TABLE VIII
RESULTS OF THE SECOND EVALUATION STAGE OF THE 2009 CIG LEG; SCORES ARE COMPUTED AS THE MEDIAN OVER EIGHT RACES

Competitor	Migrants	Buzzard Raceway	Forza	Total
Butz & Lönneker	12	11	7	30
Onieva & Pelta	8	9	12	29
Beelitz	6	4	5.5	15.5
Munoz	5	6	3.5	14.5
Cardamone	4	4.5	4	12.5
Quadflieg & Preuss	2.5	3.5	6	12
Chiu	4	2.5	4	10.5
Perez & Saez	4	3	2.5	9.5
Ebner & Thiede	0	0	0	0
Wong	0	0	0	0
Vrajitoru & Guse	0	0	0	0
Szymaniak	0	0	0	0
Bernardi et al.	0	0	0	0

TABLE IX
FINAL STANDINGS OF THE 2009 SIMULATED CAR RACING CHAMPIONSHIP

Competitor	CEC	GECCO	CIG	Total
Onieva & Pelta	22	32	29	83
Butz & Lönneker	28.5	16.5	30	75
Cardamone	20	23	12.5	55.5
Perez & Saez	16	11	9.5	36.5
Quadflieg & Preuss	19	0	12	31
Chiu	14	0	10.5	24.5
Munoz	0	8.5	14.5	23
Beelitz	0	0	15.5	15.5
Vrajitoru & Guse	0	13	0	13
Wong	0	12.5	0	12.5
Bernardi et al.	0	11	0	11
Ebner & Thiede	0	0	0	0
Szymaniak	0	0	0	0

a long timespan and to compare their controllers with the competitors of the previous editions. As a result, the performance of the submissions improved significantly over time: the winners of the 2009 championship outperform the winners of the 2008 competitions with a good margin, and are competitive with one of the best controllers provided with *TORCS* (Berniw) despite the fact that they have access to less information. In fact, while our competitors have access to a rather simple sensory model (Section IV), the controllers provided with *TORCS* have access to accurate and complete knowledge about the whole track geometry, about the physics of the car (e.g., the friction of the tires, the weight of the car, the power of the brakes, etc.), and about position and speed of the opponents.

The results of the championship showed significant improvements of the controllers submitted in terms of reliability and overtaking/opponent management. While early controllers frequently crashed and sometimes failed to get back on track (especially when faced with tracks dissimilar to those they had been trained on), the best controllers now rarely crash and always get back on track quickly. In addition, while early controllers often slammed straight into their opponents, the best controllers now keep a safe distance from opponents and choose better opportunities for overtaking. Drawing on the body of work submitted to the simulated car racing competitions and the results obtained, we make the following observations.

1) *Offline optimization works:* All the best drivers had their parameters tuned using an evolutionary algorithms (e.g., to evolve the weights of a neural network or to tune the parameters of a hand-coded controller) while hand-tuned controllers were mainly found at the bottom of the league table (Table IX).
2) *Online learning works:* Several competitors included some simple form of online learning to remember previous crash situations so as to adapt their driving style (e.g., by slowing down) in subsequent laps. These mechanisms were usually built on top of an existing controller and worked according to a subsumption-like mechanism. Whether a more sophisticated mechanism of online learning might be practical is still an open issue, as none of the competitors took such type of approach.
3) *Imitation still needs work:* Some competitors tried to develop drivers by imitating human players using forms of supervised learning. However, the performance was limited and the approach has been abandoned by most of the participants. This is coherent with the published works which show that, in car racing, imitation-based learning generally fails to produce competitive controllers (e.g., [4], [35], and [5]).

B. On the Organization

The championship has been a success both in quantitative terms (for the number of submitted controllers, of attendees at the competition sessions, of mentions in the media, etc.) and in qualitative terms (the scientific quality of the controllers was rather high as demonstrated by the several related publications [35], [30], [24], [11]). Based on their three years experience with car racing competitions, the organizers would like to draw the following considerations.

1) *It takes time:* To begin with, organizing a competition takes time. The competition software needs to be debugged, optimized, and streamlined.
2) *Make it worthwhile:* Competitions are typically run as standalone events. When faced with the announcement of a new competition, researchers have to decide whether the investment is worth and ask themselves several questions. Do they have a chance to win? Will they publish from it? Will there be another edition? Will they find enough information and support?
   We believe that *successful competitions need to be run as a series, evolving gradually*. Competitors need time to develop and tune their contributions. New competitors, who have not taken part in previous editions will get enticed by seeing and reading about a competition and join a later edition, bringing fresh ideas. Later editions of a competition will thus almost always have more and better contributions. The competition software should be as backwards compatible as possible, so that contributions to earlier editions can easily be entered in new editions of a competition. A complete break with tradition (rules and/or API) should only be made when there is a very compelling reason, such as the problem being essentially solved and not much further improvement can be expected.
3) *It is about simplicity and being open:* The competition software must provide *a very simple interface* (API) to attract as many competitors as possible. It must also provide simple examples of controllers and training algorithms, so that it should take little effort to get started with the competition. The competition software should be open to any programming language and to any operative system. In fact, some of the best controllers of this championship were written in C++, others in Java (in an earlier edition of the competition, one competitor used Perl), while our competitors developed both on Linux and Windows (unfortunately, we were not able to provide a Mac environment due to some compiling issues with *TORCS*). Finally, competition software should be open. In fact, there are no downsides to having the competition software be open source, only upsides, as competitors can help out with debugging the package, and can get an understanding of how it works faster and without asking the organizers (less effort for the latter). Accordingly, submissions should also be mandatory open sourced, as being able to inspect the winning entries increases the value of the competition for the scientific community, and also works as an extra safeguard against the unlikely possibility of cheating.
4) *It is about the Students:* In our experience, game-related competitions make for excellent student assignments, at all levels. Most of the best submissions we received involved Ph.D. students. In addition, we have been told that the single competitions have also been used as student assignments in several courses at the undergraduate and graduate levels.
5) *It is about learning:* Finally, the organizers would like to point out that this competition, in its current form, is not meant to (and cannot) measure the performance of learning algorithms. Nevertheless, it still gives researchers a good chance to prove that learning algorithms are effective, as it is in principle possible for a nonlearning algorithm to win the competition.

VIII. Outlook

The simulated car racing competitions will continue, given that there is suitable interest from the academic community. Our aim is to take the simulated car racing competition a step further, but, at the same time, to keep the API and the rules as similar as possible to the ones used in previous editions. This will allow previous competitors to enter the new competition with almost no additional effort. In particular, we plan to extend the current setup of the competition as follows. First, we will provide a more modular implementation of the sample controller to simplify as much as possible the development of an entry based on it. Second, we will extend the sensory inputs, adding also noise. Third, the effectors will be extended to include the control of the clutch. In addition, we will increase the number of laps of the race so that damage control will become a much more relevant factor. Finally, we will introduce a *warm-up stage*, where the controllers will race alone on the track before being evaluated. The aim of the warm-up stage is to encourage the application of online learning techniques that may allow the optimization of the controllers for the track before the actual evaluation takes place.

Besides the main simulated car racing competition, we think that other *forms* of simulated car racing competition might be interesting. In particular, specific *competition tracks* might focus on the game content generation, on developing an adaptive AI, and on improving the user game experience.

ACKNOWLEDGMENT

The organizers of the championship, D. Loiacono, P. L. Lanzi, and J. Togelius, would like to thank all the competitors, without whom there would be no competition or championship at all.

REFERENCES

[1] J. Togelius and S. M. Lucas, "Evolving controllers for simulated car racing," in *Proc. Congr. Evol. Comput.*, 2005, pp. 1906–1913.

[2] J. Togelius, S. M. Lucas, and R. De Nardi, "Computational intelligence in racing games," in *Advanced Intelligent Paradigms in Computer Games*, N. Baba, L. C. Jain, and H. Handa, Eds. New York: Springer-Verlag, 2007.

[3] J. Togelius, "Optimization, imitation and innovation: Computational intelligence and games," Ph.D. dissertation, Dept. Comput. Electron. Syst., Univ. Essex, Colchester, U.K., 2007.

[4] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Learning drivers for TORCS through imitation using supervised methods," in *Proc. IEEE Symp. Comput. Intell. Games*, Sep. 2009, pp. 148–155.

[5] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, "Robust player imitation using multiobjective evolution," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 652–659.

[6] A. Agapitos, J. Togelius, S. Lucas, J. Schmidhuber, and A. Konstantinidis, "Generating diverse opponents with multiobjective evolution," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec. 2008, pp. 135–142.

[7] L. Cardamone, D. Loiacono, and P. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 2622–2629.

[8] J. Togelius, S. M. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow, "The 2007 IEEE CEC simulated car racing competition," *Genetic Programm. Evolvable Mach.* 2008 [Online]. Available: http://dx.doi.org/10.1007/s10710-008-9063-0

[9] The Open Racing Car Simulator, [Online]. Available: http://torcs.sourceforge.net/

[10] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated Car Racing Championship 2009: Competition software manual," Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, Tech. Rep. 2009.04, 2009.

[11] E. Onieva, D. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the TORCS racing engine," in *Proc. IEEE Symp. Comput. Intell. Games*, 2009, pp. 256–262.

[12] L. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, pp. 338–353, 1965.

[13] R. Garcia and T. de Pedro, "Modeling a fuzzy coprocessor and its programming language," *Mathware Soft Comput.*, vol. 5, no. 2–3, pp. 167–174, 1998.

[14] M. Sugeno, "On stability of fuzzy systems expressed by fuzzy rules expressed with singletons consequents," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 2, pp. 201–224, Apr. 1999.

[15] E. Onieva, V. Milanés, J. Perez, J. Alonso, and T. de Pedro, "Soft computing techniques for autonomous driving," *Mathware Soft Comput.*, vol. 16, no. 1, pp. 45–58, 2009.

[16] J. Alonso, N. Serrano, T. de Pedro, C. González, and R. Garcia, "Optimization of an autonomous car fuzzy control system via genetic algorithms," in *Proc. Int. Workshop Genetic Fuzzy Syst.*, 2005, pp. 101–106.

[17] V. Milanés, J. E. Naranjo, C. González, J. Alonso, and T. de Pedro, "Autonomous vehicle based in cooperative GPS and inertial systems," *ROBOTICA*, vol. 26, pp. 627–633, 2008.

[18] F. Herrera, M. Lozano, and J. Verdegay, "Tuning fuzzy logic controllers by genetic algorithms," *Int. J. Approximate Reason.*, vol. 12, pp. 299–315, 1995.

[19] A. Homaifar and E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controller using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 2, pp. 129–139, Apr. 1995.

[20] E. Onieva, J. Alonso, J. Perez, V. Milanes, and T. de Pedro, "Autonomous car fuzzy control modeled by iterative genetic algorithms," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Aug. 2009, pp. 1615–1620.

[21] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press, 1984.

[22] M. V. Butz, O. Sigaud, and P. Gérard, "Anticipatory behavior: Exploiting knowledge about the future to improve current behavior," in *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems*, ser. Lecture Notes in Computer Science, M. V. Butz, O. Sigaud, and P. Gérard, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 1–10.

[23] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, pp. 159–195, 2001.

[24] M. V. Butz and T. Lönneker, "Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge," in *Proc. IEEE Symp. Comput. Intell. Games*, 2009, pp. 317–324.

[25] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," in *Proc. 11th Annu. Conf. Genetic Evol. Comput.*, New York, 2009, pp. 1179–1186.

[26] L. Cardamone, "On-line and off-line learning of driving tasks for the open racing car simulator (TORCS) using neuroevolution," M.S. thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, 2008.

[27] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, 2002.

[28] M. Verzola, "Neuroevolution of augmenting topologies applied to the open racing car simulator," M.S. thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, 2007, supervisor: Prof. P. L. Lanzi; co-supervisor: Dr. D. Loiacono.

[29] L. Zadeh, "A new direction in AI—Toward a computational theory of perceptions," *AI Mag.*, vol. 22, no. 1, pp. 73–84, 2001.

[30] D. Perez, G. Recio, Y. Saez, and P. Isasi, "Evolving a fuzzy controller for a car racing competition," in *Proc. IEEE Symp. Comput. Intell. Games*, Sept. 2009, pp. 263–270.

[31] J. Quinlan, *C4.5: Programs for Machine Learning*, ser. Machine Learning, 1st ed. San Mateo, CA: Morgan Kaufmann, Jan. 1993.

[32] S. A. K. Deb, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2000.

[33] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, ser. Data Management Systems, 2nd ed. San Mateo: Morgan Kaufmann, Jun. 2005.

[34] D. Loiacono, J. Togelius, P. Lanzi, L. Kinnaird-Heether, S. Lucas, M. Simmerson, D. Perez, R. Reynolds, and Y. Saez, "The WCCI 2008 Simulated Car Racing Competition," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec. 2008, pp. 119–126.

[35] J. Munoz, G. Gutierrez, and A. Sanchis, "Controller for TORCS created by imitation," in *Proc. IEEE Symp. Comput. Intell. Games*, Sep. 2009, pp. 271–278.

[36] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proc. Conf. Companion Genetic Evol. Comput.*, New York, 2008, pp. 2701–2726.

[37] A Java-Based Evolutionary Computation Research System, [Online]. Available: http://cs.gmu.edu/eclab/projects/ecj/

[38] M. Ebner and T. Tiede, "Evolving driving controllers using genetic programming," in *Proc. IEEE Symp. Comput. Intell. Games*, Sep. 2009, pp. 279–286.

Daniele Loiacono was born in Lecco, Italy, in 1980. He graduated *cum laude* in computer engineering from Politecnico di Milano, Milan, Italy, in 2004 and received the Ph.D. in computer engineering from the Department of Electronics and Information of Politecnico di Milano in 2008.

Currently, he is a Postdoctoral Researcher at the Department of Electronics and Information, Politecnico di Milano. His main research interests include machine learning, evolutionary computation, and computational intelligence in games.

Pier Luca Lanzi was born in Turin, Italy, in 1967. He received the Laurea degree in computer science from the Università degli Studi di Udine, Udine, Italy, in 1994 and the Ph.D. degree in computer and automation engineering from the Politecnico di Milano, Milan, Italy, in 1999.

Currently, he is an Associate Professor at the Department of Electronics and Information, Politecnico di Milano. His research areas include evolutionary computation, reinforcement learning, and machine learning. He is interested in applications to data mining and computer games.

Dr. Lanzi is a member of the editorial board of the *Evolutionary Computation Journal*, the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, and *Evolutionary Intelligence*. He is also the Editor-in-Chief of the *SIGEVOlution*, the newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation.

Julian Togelius received the B.A. degree in philosophy from Lund University, Lund, Sweden, in 2002, the M.Sc. degree in evolutionary and adaptive systems from University of Sussex, Sussex, U.K., in 2003, and the Ph.D. degree in computer science from the University of Essex, Colchester, U.K., in 2007.

Currently, he is an Assistant Professor at the IT University of Copenhagen (ITU), Copenhagen, Denmark. Before joining ITU, he was a Postdoctoral Researcher at the Dalle Molle Institute for Artificial Intelligence (IDSIA), Lugano, Switzerland. His research interests include applications of computational intelligence in games, procedural content generation, automatic game design, evolutionary computation, and reinforcement learning.

Dr. Togelius is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES and a Vice Chair of the IEEE Computational Intelligence Society Games Technical Committee.

Enrique Onieva received the B.E. degree in computer science engineering and the M.E. degree in soft computing and intelligent systems from the University of Granada, Granada, Spain, in 2006, and 2008, respectively.

Since 2007, he has been with the Industrial Computer Science Department, Industrial Automation Institute, Spanish Council for Scientific Research, Madrid, Spain.

David A. Pelta received a computer science degree from the National University of La Plata, La Plata, Argentina, in 1998 and the Ph.D. degree in computer science from the University of Granada, Granada, Spain, in 2002.

Currently, he is a Professor at the Department of Computer Science and AI, University of Granada. Among his research interests are soft computing techniques, cooperative strategies for optimization, adversarial reasoning, and self-adaptive systems. He is a member of the Models of Decision and Optimization Research Group. He is involved in research projects funded by the Spanish Government and the European Community. He has published several journal papers, coedited three books and five special issues on relevant journals.

Dr. Pelta serves on the editorial board of the *Memetic Computing* journal and acts as reviewer for many journals, including *Bioinformatics*, *Soft Computing*, *Swarm Intelligence*, etc.

Martin V. Butz received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, in 2004. His dissertation "Rule-based evolutionary online learning systems" puts forward a modular, facet-wise system analysis for learning classifier systems (LCSs) and analyzes and enhances the XCS classifier system.

He has been working at the Department of Cognitive Psychology III, University of Würzburg, Würzburg, Germany, on various interdisciplinary cognitive modeling projects. In October 2007, he founded his own cognitive systems laboratory: Cognitive Bodyspaces: Learning and Behavior (COBOSLAB), funded by the German research foundation under the Emmy Noether framework. His participation in the racing car competition shows that anticipatory behavior principles are also applicable to other areas of artificial intelligence, including computational intelligence in games.

Thies D. Lönnecker started studying computer science and engineering at the Hamburg University of Technology, Hamburg, Germany, in 2001 and switched to the University of Würzburg, Würzburg, Germany, in 2005, for studying computer science and linguistics.

He has been working as a tutor at the Department of German Studies in 2007 and 2008. Due to his knowledge in the field of cognitive systems and as part of the preparation for his diploma with the focus on artificial intelligence, he designed a controller for the "2009 Simulated Car Racing Championship" at COBOSLAB.

Luigi Cardamone was born in Ispica (RG), Italy, in 1984. He received the first level degree *cum laude* and the second level degree *cum laude* in computer engineering from the Politecnico di Milano, Milan, Italy, in 2006 and 2008, respectively. Currently, he is working towards the Ph.D. degree in computer engineering at the Department of Electronics and Information, Politecnico di Milano.

His research area is in computational intelligence and games.

Diego Perez was born in Madrid, Spain, in 1983. He graduated in 2007 in computer science and received the M.S. degree in the same field from Carlos III University, Madrid, Spain, in 2008.

His research is centered in the application of evolutionary computation to games, focusing on problems of classification, optimization, and machine learning, and participating in several game competitions held at the IEEE Congress on Evolutionary Computation. He has experience in the videogame industry and is currently working at the National Digital Research Center, Dublin, Ireland, developing artificial intelligence tools that can be applied to the latest industry videogames.

Yago Sáez received the computer engineering degree from the Universidad Pontificia de Salamanca, Salamanca, Spain, in 1999 and the Ph.D. degree in computer science from the Universidad Politecnica de Madrid, Madrid, Spain, in 2005.

Currently, he is Vice-Head of the Computer Science Department, Carlos III University of Madrid, Madrid, Spain, where he is an Associate Professor. He is part of the Evolutionary Computation, Neural Networks and Artificial Intelligence group (EVANNAI) and member of the IEEE Computational Finance and Economics Technical Committee. His main research area encompasses the evolutionary computation, the computational economic and finance applications, and the optimization by means of metaheuristics.

Mike Preuss was born in Ahlen/Westfalen, Germany, in 1969. He received the Diploma degree in computer science from the Technische Universität Dortmund (TU Dortmund), Dortmund, Germany, in 1998.

Currently, he is a Research Associate at the Computer Science Department, TU Dortmund (since 2000). His research interests focus on the field of evolutionary algorithms (EA) for real-valued problems, namely on multimodal and multiobjective niching. His mission is to further develop the experimental methodology for nondeterministic optimization algorithms by means of tuning, adaptability measures, and other experimental analysis techniques. These are essential for the design of specific EA-based algorithms for industrial optimization in various engineering domains, e.g., thermodynamics and ship propulsion technology. As Chair of the IEEE Computational Intelligence Society task force on real-time strategy games, he is active in designing authentic AI components for believable opponents and increased player satisfaction.

Jan Quadflieg was born in Schwelm, Germany, in 1980. In 2000, he began to study computer science at the Technische Universität Dortmund (TU Dortmund), Dortmund, Germany, where he will receive the Diploma degree in 2010.

From 2006 to 2007, he was a Student Assistant at the research group Innovative Factory Systems (IFS), German Research Center for Artificial Intelligence, Kaiserslautern, Germany. His research interests include computer games, especially real-time 3-D graphics and the application of computational intelligence methods, and the visualization of high-dimensional data sets.