A Finite State Machine Controller for the Simulated Car Racing Championship

Bruno H. F. Macedo*, Gabriel F. P. Araujo*, Gabriel S. Silva*,
Matheus C. Crestani*, Yuri B. Galli* and Guilherme N. Ramos*†
*University of Brasília †gnramos@unb.br

***Abstract*—In this paper, a solution is provided to a driver in the The Open Racing Car Simulator (TORCS). The TORCS plus the SCR provides a interface capable of being developed a driver. This solution have a FSM to choose how to drive a piece of track. And it was tested the parameters changing of the states in the FSM. Therefore, it can see how parameters affect the performance car, the results demonstrate that it have a lot of things to learn, but the are in the right way.**

***Index Terms*—finite state machines, computer games, Simulated Car Racing Championship, artificial intelligence**

I. Introduction

DIGITAL games provide a great test bed for experimentation and study of Artificial Intelligence (AI), and there has been a growing interest in applying AI in them, regardless of genre [1]. Some applications include controlling non-playable characters [2], path planning [3], human pose recognition [4], and others.

Electronic games also present a well defined environment, which may simulate extremely complex situations such as flying an airplane or controlling a car [6]. One such simulator is TORCS, *The Open Racing Car Simulator* [5] whose input and output are constrained by the software's characteristics, providing an ideal benchmark for comparing AI solutions for creating car controllers.

This paper is organized as follows: Section 2 presents the details on TORCS and the Simulated Car Racing Championship and some background on finite state machines, Section 3 describes the implementation of the FSMDriver, Section 4 presents initial experimental results, and Section 5 provides concluding remarks.

II. Background

Drive a car is interesting problem, if human component was removed, because it is not a trivial problem and humans can do easily. There are a many visions to solve this puzzle, for example, approach human-human. Thrun et Al [**?**] show that it is possible to solve this without manual intervention with real car.

But the problem show in this paper is less complex which a controller of real car. Here, expose a virtual problem, run and win a racing car game without a player. In the literature, there are works using many types of solution to drive a autonomous car. This solutions comprehended famous artificial intelligence approaches such as Supervised Methods with NEAT, Finite State Machine with Fuzzy Logic or Tracking and Planning ahead in a racing.

The literature show that is not a good idea choose supervised method in a game-AI. Nevertheless, [**?**] using high-level information about the current status of the car and the road ahead was a possible around this trouble. Using too NEAT to replace the player, he demonstrated it is possible develop drivers with performances that in some cases are only 15% lower than the performance of the fastest driver available in TORCS.

III. The Simulated Car Racing Championship

The Simulated Car Racing Championship, or simply SCR, [6] is a well-known event comprising three sequential competitions held in association with important names such as *IEEE*, and present in ackowledged conferences such as the *Congress on Evolutionary Computation* (CEC), *ACM Genetic and Evolutionary Computational Conference* (GECCO) and the *Symposium on Computational Intelligence and Games* (CIG). This competition held at such major conferences in the fields of Evolutionary Computation and Computational Intelligence - and their standard relations to Gaming in general - uses *The Open Car Racing Simulator* (TORCS) as its pillar for all activities of development and analysis. This simulator is state-of-art in the car racing game sphere, for it provides not only an outstanding method of comparing algorithms in the topics cited, but also the full 3D visualization of every process, a life-like physics engine containing movement and dynamic aspects in a highly-detailed manner, such as fuel usage, damage received to the car, wheel traction and so on.

A. The Open Car Racing Simulator

TORCS is a platform focused on the establishment of ground rules towards the comparison of algorithms concerning *programming* and *AI*. The purpose is to create intelligent controllers to simulate real-life situations of racing competitions, and to do so, the Formula 1 score/awarding system is used as unbiased criteria.
The interface provides with a diversified set of sensors, such as the current position of the car in the track, its acceleration and brake values, and so on. Utilizing this sensors is the method with which the users communicate with their created pilots and the environment, and it is of their responsibility to program the entire behavior of this pilot, in whichever programming language of desire.

TORCS serves both as a research platform for AI on racing development and as an ordinary car racing game. The main reason why TORCS is widespread in the AI gaming universe is its portability; it runs on all *Linux*'s architectures, *FreeBSD*, *OpenSolaris*, *MacOSX* and both 32 and 64 bits *Windows*. TORCS started some years ago to have its own competitions, where groups of development can submit their pilots in order to compete with others, and the goal of FSMDriver - the pilot of this publication's topic - is to be submitted to one of such competitions.

B. Algorithms implemented in the Simulated Car Racing Championship

Works submitted to TORCS have a wide range of variety, going from sophisticated heuristic designs to completely mathematical and statistical approaches. The controller that won the 2009 Simulated Car Racing Championship, [**?**] perhaps one of the most important editions realized so far, was created by *Enrique Onieva* and *David A. Pelta*and it consists on a simple set of controllers in an *modular configuration*. Although this competitors won the competition, a modular approach for controllers was weighed and the effort to implement it would be increasingly troublesome as more problems were introduced and the whole picture became more complex; also, reworking and debugging such an architecture requires changing perhaps all the other modules not involved in the change, as the individual modules are not entirely independent. Another well-rewarded method was the *anticipatory behavior*; such an implementation is even more complex than the modular configuration, as it envolves a remarkable amount of relations between sensor entries, and has to provide a considerable amount of outputs, and the analysis previous to the proper assembly of the controller requires a deeper understanding of the operation of the physical model in question. As a last example of previous works submitted to Simulated Car Racing Championships, *Neural Networks* were chosen; using Neural Networks is a general approach that requires possibly the lowest domain knowledge of all, and provides a satisfactory result in comparison, but its implementation is also quite complex, and involves thinking of an extense number of possibilities for sensor inputs.
Given the overall look on the previous controllers that acquired a high performance on the competitions, the deficiencies and the missing perks were all weighed, and the solution came in the form of a *Finite State Machine*, whose characteristics cover: the lack of simplicity of other methods, the implementation difficulties they presented and the independence of the parts in which the whole problem is divided. Such properties of a Finite State Machine are highlighted in Section 3.A.

C. Finite State Machines

Finite State Machines (FSM), *Finite-State Automaton*, or even the simple *State Machine*, are designations of the same model. They all describe a mathematical model of computation used in computer programs, such as this one, and also in sequential logic circuits. FSM has a typical *abstract machine* behavior, with a limited number of states, and, by definition, can be only and exclusively in one of these states at a time. The state in which the machine is in a present/defined moment will be hereby called the *current state*, as a matter of convenience. The machine changes its current state when triggered by a pre-defined event, also hereby called a *transition*.

Thus, a FSM is a model defined by states, which the *current state* is changed every time a *transition* defined by the designated conditions is triggered. According to Mat Buckland's definition published inside *Programming Game AI by Example* [11], "*A finite state machine is a device, or a model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine can only be in one state at any moment in time.*"

IV. The FSM Driver

The method of choice to submit the pilot to future TORCS competitions is, naturally, a *Finite State Machine*. A brief explanation of the behavior of this method was presented in section 2.C, for more information[2], FSMs are widespread over the academic knowledge in books and in the web.

A race inside TORCS has some peculiarities perhaps only visible for those who possess a deeper understanding on the behavior of a real car, however, a layperson can absolutely develop a pilot in the environment, given the easy features the interface provides. The communication between programmer and car is the main aspect that led the choice of a FSM for the pilot. FSMs have been one of the favorite instruments for AI gaming, and they are present in almost every game due to many reasons as follows:

- **Readiness**: A FSM is quickly and simply implemented, in almost every one of its forms;
- **Modularity**: Debugging and reworking a FSM become catalyzed processes due to the abstraction of its structure, that is, its behavior can be divided into smaller, independent parts, which can be treated separately in a way that reduces labor;
- **Low computational complexity**: A FSM would hardly ever require great amounts of processor time, and that is because its operation mainly follows but hard-coded rules;
- **Intuitive behavior**: Analyzing a FSM is an easy process because human minds are often used to categorizing situations and conditions of conduct. This characteristic is both a good employment for debugging in real time and for turning the pilot's behavior simple enough so that a common person, i.e. non-programmer, could identify mistakes and spectate a race understanding the development of the pilot;
- **Flexibility**: A fundamental quirk of a FSM, one which makes the process of expanding the scope of the structure easier in a level hardly other choice of method would provide. If a new situation occurs, one that was not

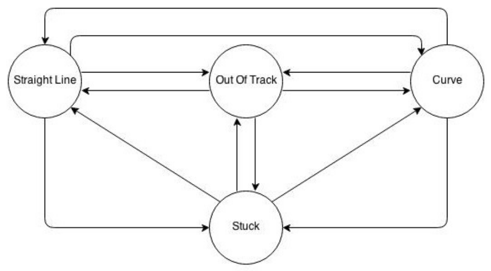

Fig. 1. State transition diagram.

foreseen by the states defined and also one that could not be inserted in any of them, the mere creation of a new state would solve the matter, in about no time and almost without interference in the other parts of the whole.

The division of states for the FSMDriver itself are in agreement with the advantages of a FSM, meaning that the choice of which states to be used relied on the human intuition of the development of a car in operation. The states designed are:

- **Straight line**: The state in which the pilot commands the car to full throttle and maximum speed/acceleration ahead, and occurs when the car is located in a situation that sees no turning nearby;
- **Curve**: Situations focused on the steering of the car, that is, changing its direction of movement in order to keep it inside the delimitations of the proper racing space and also to avoid crashing the car into an obstacle;
- **Out of track**: When the previous states fail to prevent the car from entering an undesired situation such as going outside of the track, where conditions such as friction are much worse for the car than inside the track, this state takes control of the car and tries to return it to the race;
- **Stuck**: In a worst case scenario, when the car is barely moving, drastic measures need to be taken, such as reversing the car and steering it out of an obstacle. The appearence of such events reduce largely the performance of the pilot in the race.

In other words, the pilot will encounter four situations in a race, two of them are desired, whereas the other two are emergency actions. While the car is either on a *straight line* or on a *curve*, the behavior is normal; but when the criteria defined to manipulate the pilot into entering those states fails to keep it racing, and the car goes *outside of the track* or gets *stuck*, the car encounters troubles and loses performance. That is why the accurate definition of the requirements the pilot has to fulfill to be in each of these states needs to be thought thoroughly for good and competitive results.

V. Methodology

Complete the architecture of the controller, it's time to find a goods parameters for its states.

At the initial moment it was chosen to begin with a exhaustive search, for each and every parameter an initial and maximum value were defined and at each iteration a delta parameter was added. The values of initial, maximum and delta was arbitrary chosen.

Each race begins with the car stopped and has three laps. The first battery test was in just one track, but it's clearly necessary to test with a large number of track to produce a general parameter for the pilot.

When the tests reaches the end and the results are generated, it is locate the parameters that makes the pilot run with the minimum total time. At the time, the total time it's the only demand to make a distinction os results.

The very first results emphasize this controller achievements: a better performance in more simple tracks, like oval ones, rather than in those with a higher degree of elaboration, like road or dirty tracks.

VI. Experimental Results

When racing in oval tracks, particularly in B-speedway, the FSMDriver outperforms some of the controllers provided by the TORCS distribution. The FSMDriver raced against considered average drivers and performed very well, winning even if it started behind them.

TABLE I
Results of a 3 Laps Race Held in B-Speedway against Berniw 10

Driver	Total Time	Damage	Top Speed
FSMDriver	02:34:95	1	299
Berniw 10	02:40:59	8	282

TABLE II
Results of a 3 Laps Race Held in B-Speedway against Olethros 10

Driver	Total Time	Damage	Top Speed
FSMDriver	02:34:88	1	299
Olethros 10	02:47:38	5	282

TABLE III
Results of a 3 Laps Race Held in B-Speedway against Iliaw 10

Driver	Total Time	Damage	Top Speed
FSMDriver	02:34:90	1	299
Illaw 10	02:39:82	0	283

In the other hand, when facing considered good drivers the controller was defeated. However, if more than one driver race against FSMDriver at the same time, it will benefit from the crashes and finish in better positions.

The damage taken can be significantly reduced by implementing an enemy avoidance and overtaking behavior,

Although, when it comes to road tracks, especially the ones filled with critical curves combinations, the controller faces several problems at staying between the track boundaries. These troubles are mainly caused by entering curves with high speed values, fact that reinforces the idea of an approaching curve state, to deal with speed reductions and angle corrections. In addition, the constants used in transition process do

TABLE IV
RESULTS OF A 3 LAPS RACE HELD IN CG SPEEDWAY 1 AGAINST 6 CONSIDERED GOOD DRIVERS

Driver	**Total Time**	**Damage**	**Top Speed**
Inferno 3	02:30:92	0	300
Olethros 3	02:33:19	210	317
bt 3	02:33:23	0	303
FSMDriver	02:33:30	315	309
Berniw 3	02:39:52	0	312
Iliaw 3	02:40:62	1635 0	314
Tita 3	02:52:15	1794	300

not represent the optimal values for that role, an evolutionary method needs to be implemented in order to achieve those values. Moreover, oval tracks barely requires a state transition, in the overwhelming majority of time the car fits in the straight state. This lack of transitions and accentuated curves improves significantly the gain in performance.

A. *Parameter changing*

Each state of the FSM uses specific parameters for ruling the state behavior. The transition function as well uses constants to decide whenever state must change. At the beginning those parameter were imported from the Simple Driver provided by Danielle Loiacono, as expected they did not lead to satisfying results when compared to the drivers available in TORCS.

For initial improvements the transition parameters have been chosen. This function mainly uses four constants:

- MAX_SPEED_DIST, the maximum distance for recognizing a curve.
- LEFT_EDGE, track position value that indicate the left boundary of the track.
- RIGHT_EDGE, track position value that indicate the right boundary of the track
- STUCK_TICKS, number of ticks required to leave the stuck state.

TABLE V
PARAMETERS USED FOR ENHANCEMENT

p	p_0	p_{max}	Δp
MSD	20	200	5
LE	-1	-0.8	0.1
RE	1	0.8	-0.1
ST	30	300	10

In the end 8658 combinations were generated and properly tested in *CG Speedway 1* for three laps. The best result generated used the following parameters: MAX_SPEED_DIST: 20, LEFT_EDGE: -0.8, RIGHT_EDGE: 1, STUCK_TICKS: 110.

This approach demands a lot of time and space since all combinations are generated and executed. For future evolutions a more robust method, like genetic algorithm or neural networks needs to be applied in order to provide reliable results.

TABLE VI
COMPARISON AMONG LUPO BIANCO, WHO ACHIEVED THE BEST RESULTS IN CG SPEEDWAY 1, BERNIW 3 AND THE FSMDRIVER

Driver	**Total Time**	**Best Lap**	**Top Speed**
Lupo Bianco	01:50:12	00:35:22	262
Berniw 3	02:07:28	00:40:97	247
FSMDriver	02:34:49	00:50:12	242

VII. CONCLUSION AND FUTURE WORKS

In this article it was presented a Finite State Machine approach, in its early days, to the problem of simulated car racing at the TORCS.

At this stage the pilot showed can win just in few tracks and just of some bots available at the package TORCS. The code developed aim to win some of the competitions at the conference it's necessary, at least, to be better than the bots on average of the various tracks, so there's a lot a work to do to achieve this goal.

It's also possible to run with the racers of the past simulated car racing, meaning that you know when you are ready to a real competition at the next conference.

The creation of a state to deal with opponents it's the next step in effort to take a better performance at racing, without this it's necessary to count with some luck at the moment of interaction with some opponent.

The high damage received at the tests show us that it need some improvement. A close look show that at the moment of transition from a straight line to a curve the car lost the control and went out of track, and eventually, the car crashes somewhere and get a damage. One possible way to deal with this problem is an insertion of another state, that prepare the car to a curve, adjusting the speed and location above the axis of track. A filter on the brake can be a good solution in order to minimaze the speed without allowing the car to skid on the track.

To summarize there's a vast open way to improvements, maybe using some evolutionary algorithm over the exhaustive search applied here to find a good combination of parameters.A change of paradigm also could be good, for instance, change the PID controller to some fuzzy logic. The advantage of a simulator is that all the questions about performance can be solved with the simulation.

REFERENCES

[1] S. M. Lucas, "Computational intelligence and games: Challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11633-008-0045-8

[2] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 653–668, 2005. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1545941

[3] L. G. de Freitas, L. M. Reffatti, I. R. de Sousa, A. C. Cardoso, C. D. Castanho, R. Bonifácio, and G. N. Ramos, "Gear2d: an extensible component-based game engine," in *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2012, pp. 81–88. [Online]. Available: http://dl.acm.org/citation.cfm?id=2282357

[4] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 1297–1304. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5995316

[5] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 131–147, 2010.

[6] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, The Open Racing Car Simulator," http://www.torcs.org, 2014.

[7] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Winning the darpa grand challenge," *Journal of Field Robotics*, 2006.

[8] P. L. L. L. Cardamone, D. Loiacono, "Learning drivers for torcs through imitation using supervised methods," *Proc. IEEE Symp. Comput. Intell. Games*, 2009.

[9] M. Buckland, *Programming game AI by example*. Plano, Texas: Wordware Pub, 2005.