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*—Electronic games can simulate extremely complex situations and be used for benchmark tests for specialized software. TORCS simulates a very realistic environment, and is ideal for comparing for autonomous car controllers and, consequently, the machine learning techniques applied in generating such controllers. This paper presents an AI approach to efficiently generating a car controller based on a finite state machine. Experimental results show how the controller's parameters are selected and their effect on its performance.**

***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 [**?**]. Some applications include controlling non-playable characters [**?**], path planning [**?**], human pose recognition [**?**], 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. One such simulator is *The Open Racing Car Simulator* (TORCS) [**?**], whose input and output are constrained by the software's characteristics, providing an ideal benchmark for comparing AI solutions for creating car controllers.

Due to the complexity of the task, most solutions attempt to develop an efficient controller by evolving it completely, that is, by considering its performance on the whole track. This is a very complex task, and in an attempt to reduce this complexity, this works proposes the FSMDriver, a finite state machine based controller that divides the track into different types of stretches and evolves a state for handling each one.

The practical applications of such controllers in autonomous vehicles are enormous, and have been the subject of intense research for some years. For example, the DARPA[1] has offered substantial prizes to winners of robotic challenges, the most famous being the Grand Challenge race won by Thrun et al. [**?**]
.

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

[1]http://www.darpa.mil

II. Background

The Simulated Car Racing Championship (SCR) is a well-known event comprising three sequential competitions held in association with *IEEE*, and present in well known 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) [**?**]. It provides an opportunity for the scientific community to perform a straightforward comparison among different approaches in complex environments containing multiple continuous variables [**?**]. For this, it uses *The Open Car Racing Simulator*, a state-of-the art car simulator.

A. The Open Car Racing Simulator

TORCS provides a very customizable environment, sophisticated 3D graphics, and - most importantly - a powerful physics simulation platform [**?**]. Its physics engine considers static and dynamic aspects, such as fuel usage, damage received, wheel traction, and others, in a very detailed way. Thus, it enables a very complex environment for testing AI techniques.

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 has its own online community, which started its own racing competitions some years ago [**?**], where developers can submit their controllers to compete with each other.

B. The Simulated Car Racing Championship

The SCR contest provides a standard measure for TORCS controllers by defining strict rules, based on the Formula 1 score system [**?**]. It improves on the practical applications aspects by limiting the controller's knowledge of the system to the information provided by its inputs and its actions by its outputs, much like an actual autonomous vehicle.

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 [**?**]. The information provided by them defines the controllers perception of its environment (track, other controllers, etc.), and the idea is to program a behavior that results in the best possible race.

C. SCR Contestants

Works submitted to SCR 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 Onieva et al. [**?**], and it consists of a simple set of controllers in an *modular configuration*.

Although this controller won the competition, the authors noted that it evolved into a fairly complex model and noted that a modular approach considerably reduce the effort to implement a controller. Refactoring and debugging such a complex architecture requires changing code that is not directly related to the desired change in behavior.

Another interesting controller used artificial neural networks [**?**], a more general approach which required little domain knowledge and provided a satisfactory result. Also in this case, the implementation is quite complex and the final result manages connections between all inputs and outputs.

In order to reduce the effort in developing a controller, a more modular approach, such as a *finite state machine*, is desirable.

D. Finite State Machines

A *Finite State Machines* (FSM) designates the model of a device, which has a finite number of states it can be in at a given moment, and which and can operate on input to either transition from one state to another or to cause an output to take place [**?**]. The machine can only be in one of these states at a given time.

FSMs have been widely used in AI applications in games [**?**], due its inherent characteristics:

- **Simplicity**: this model can be quickly implemented in many ways.
- **Flexibility**: new states can be easily incorporated into the model.
- **Modularity**: states can be independently developed in parallel.
- **Focused debugging**: since FSMs can only be in one state at a time, debugging/testing is limited to the state in question.
- **Small computational overhead**: states are usually implemented with hard-coded rules concerning a specific situation, which in turn demands small amounts of processor time.
- **Intuitive behavior**: analyzing a FSM is an easy process for humans because we are always intuitively categorizing situations and behaviors, just like a FSM working.

III. The FSM Driver

The proposal for controlling the car is a *Finite State Machine*, which enables a modular development, ideal for teamwork, and separates behaviors, reducing the effort necessary to evolve a complete controller by breaking this problem into evolving each state.

From the intuitive perspective of a human driver in a car race competition, there is a clearly defined desirable behavior: to *drive faster than the competitors*. The simulation account for different environment conditions, therefore a car's performance is always better, everything else being the same, when it is within track limits than outside. So this rather complex behavior which can be broken into specific behaviors: *drive as fast as possible on the track* and, in case something goes wrong, *get back on the track as fast as possible*.

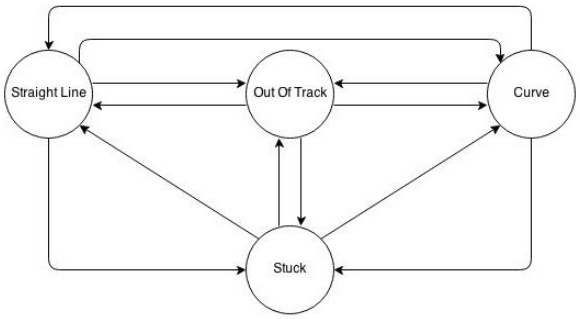

Fig. 1. State transition diagram.

Driving as fast as possible on a straight line is simple, but a different behavior than doing it in a curve, thus the state can be further divided into *drive drive as fast as possible in a straight line* and *make the curve as fast as possible*. Finally, because racing usually implies being able to move forward, a behavior for handling situations when that is not possible is necessary.

Therefore, FSMDriver's initial configuration has four states, as illustrated in Fig. **??**, described as follows:

- **Straight Line**: the controller commands the car to full throttle with maximum acceleration straight ahead, adjusting the steering to stay in the middle of the track.
- **Curve**: the controller handles curves based on the position of the car and the angle between the direction of the axes of the car and the track axis, adjusting the steering, gear, and clutch actuators to keep it moving as fast as possible within track limits while trying to avoid damage (by crashing into another car or wall).
- **Out of Track**: the controller tries to return to the track as fast as possible by steering in an arbitrary angle (with relation to the track's axis of track). Due to low friction outside of the track, accelerating and braking are proportional to the sideway speed of the car to minimize skidding.
- **Stuck**: the controller takes drastic measures to try to recover, using reverse gear and hard steering to try to get back on track in this worst case scenario.

There are two states for usual on track regular on track situations and two for off track ones. Only one of these states will be in control at a time, but all have to work together to handle the all possible scenarios and each has to be properly configured to achieve competitive results.

The complete behavior is achieved by transitioning between states, this is implemented in a `transition` function that uses specific parameters to decide when the current state needs to be changed (and which state to change to). This function operates on four constant parameters:

MSD, the maximum speed distance, which defines the maximum value for the frontal sensor input while racing at full speed;

LE, the track's left edge boundary, which defines the area available for moving the car left;

RE, the track's right edge boundary, which defines the area available for moving the car right;

ST, the number of stuck ticks, required by the controller for identifying how long it has not been moving fast enough and, thus, concluding it is stuck.

`transition` analyses the car's inputs to define if there has to be a change in state on every simulation tick. The first procedure is to check if the car is **Stuck**, that is, if it is has a very low speed (obviously, ignoring the beginning of the race when all cars ares stopped), then a counter is incremented and its value compared to the ST threshold.

In case the car is not stuck, `transition` will define the current state by analyzing its position in the track. It will be **Out of Track** if it is out of the range defined by LE and RE. Otherwise, it is considered on track and can be either on a **Straight Line**, if the frontal distance sensor's input is larger than MSD or when it is larger than the input of the adjacent side sensors, or on a **Curve** otherwise.

IV. Experimental Results

For initial evaluation of FSMDriver, only changes in the transition function's parameters were tested, with a simple iteration over arbitrary values for all. At each iteration, only one parameter value was changed. Table **??** shows the parameters, their initial (V_I) and final (V_F) values, and the step (Δ) taken in each iteration.

TABLE I
Transition Parameters

	V_I	V_F	Δ
MSD	20	200	5
LE	-1	-0.8	0.1
RE	0.8	1	0.1
ST	30	300	10

For each of the over 8500 configurations, experiments consisted of 3 laps, in the B-Speedway and CG-Speedway 1 tracks (see Fig. **??**), each race starting with one car fully stopped. B-Speedway provides the simplest track for validation, while CG-Speedway 1 has more characteristics of usual race tracks, such as sharp curves for either side, and provides a more general idea of the FSMDriver's behavior.

Results are evaluated by total race time, smaller being better. The parameter values for the fastest races are presented in Table **??**. It is interesting to note that many configurations yielded the same results, and that **Straigh Line**'s steering control is sufficient for it to run on wide curves, there no transitions were triggered in B-Speedway's tests.

To further analyze FSMDriver's behavior, it is compared to default controllers provided by the TORCS distribution. One of the maintainers of TORCS, Bernhard Wymann[2], has

[2]www.berniw.org

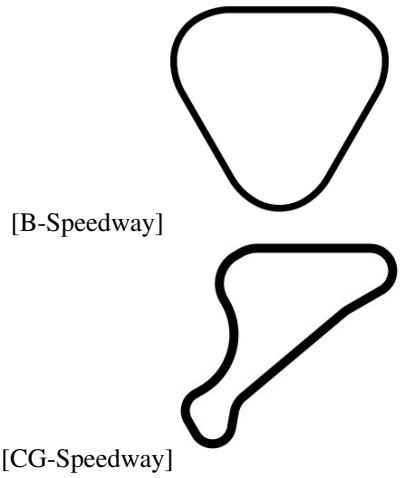

Fig. 2. FSMDriver test tracks.

TABLE II
Best Configuration

	MSD	LE	RE	ST
B-Speedway	20	-0.8	1	110
CG-Speedway 1	20	-0.8	1	110

created a wide range of controllers with diverse characteristics, from average racing behavior like *berniw3* to one of the best controllers available *berniw1*. In this test, controllers had a 3 lap race (by themselves) and total time was evaluated. Results are shown in Tables **??** and **??**.

TABLE III
3 Laps in B-Speedway

Driver	Total Time	Damage	Top Speed
Berniw 1	02:25:65	6	321
Berniw 2	02:25:65	6	321
Berniw 7	02:27:20	0	308
Berniw 6	02:27:46	0	309
Berniw 8	02:27:75	0	307
Berniw 9	02:28:10	0	306
Berniw 5	02:28:58	2	307
Berniw 4	02:29:42	3	303
Berniw 3	02:31:38	0	301
FSMDriver	02:34:95	1	299
Berniw 10	02:40:70	8	282

As expected, this initial version of FSMDriver does not have the outstanding behavior of *berniw1*, but its results show that the proposal does have acceptable performance, and future tweaking of FSMDriver's states parameters may lead to better results. It is also important to note that the TORCS controllers have a different implementation, one that has access to the simulator's inner data, while FSMDriver's behavior is defined exclusively by its available sensors.

After analyzing FSMDriver's performance in CG-Speedway 1, it was clear from the amount of damage that the problem was that the controller was consistently going out of the track and hitting the wall. After further inspection, this was found

TABLE IV
3 LAPS IN CG SPEEDWAY 1

Driver	Total Time	Damage	Top Speed
Berniw 8	02:07:17	0	246
Berniw 6	02:07:18	0	248
Berniw 4	02:07:18	0	246
Berniw 7	02:07:20	0	250
Berniw 5	02:07:28	0	246
Berniw 3	02:07:28	0	247
Berniw 9	02:07:58	0	247
Berniw 1	02:12:80	1	252
Berniw 2	02:12:80	1	252
Berniw 10	02:21:54	0	230
FSMDriver	02:34:49	3337	242

to happen after stretches of straight lines, when FSMDriver would accelerate as much as possible and enter a curve going too fast for proper steering.

This indicates the need for another state, **Approaching Curve**, in which FSMDriver would adjust between current behaviors and enter the curve in a better condition to make it as fast as possible.

V. CONCLUSION AND FUTURE WORKS

Digital games provide an excellent opportunity to experiment and study the different applications of Artificial Intelligence, such as controlling an autonomous vehicle. The The Simulated Car Racing Championship is used as a standard test for such controllers, allowing different solutions compared in a straightforward manner.

This work presents the early stages of FSMDriver, a finite state machine approach to developing a controller. The goal is to eventually compete in SCR by improving the controller's behavior though machine learning techniques. This first version considers four states for defining its racing behavior: **Straight Line**, **Curve**, **Out of Track**, and **Stuck**. Overall, the results obtained in this world provided very valuable information to guide future developments of the proposal.

Initial experimental results showed that the approach is feasible, as FSMDriver can successfully complete races and even be faster than some of the available TORCS controllers. They also showed that adjusting configuration parameters lead to improved behavior, further efforts using machine learning techniques to improve this are being pursued.

Comparison to existing controllers showed that FDSM-Driver's first version has average behavior, and there is much work to be done. To this end, future versions will consider not only the parameters that define transitions between states, but also the states' internal parameters.

Analysis of race conditions indicated an immediate need for a new state to handle between a straight stretch of track and a curve. Because experimental results showed extreme damage to the car due to crashing to the wall, it is likely that a state to *avoid collisions* will be necessary. This could also be used to handle adversary cars.