Monte Carlo Tree Search for Simulated Car Racing

Jacob Fischer[1], Nikolaj Falsted[1], Mathias Vielwerth[1],
Julian Togelius[2], and Sebastian Risi[1]
[1]Center for Computer Games Research, IT University of Copenhagen, Denmark
[2]Department of Computer Science and Engineering, New York University, NY, USA
jaco@itu.dk, nifa@itu.dk, mvie@itu.dk, julian@togelius.com, sebr@itu.dk

ABSTRACT

Monte Carlo Tree Search (MCTS) has recently seen considerable success in playing certain types of games, most of which are discrete, fully observable zero-sum games. Consequently there is currently considerable interest within the research community in investigating what other games this algorithm might play well, and how it can be modified to achieve this. In this paper, we investigate the application of MCTS to simulated car racing, in particular the open-source racing game TORCS. The presented approach is based on the development of an efficient forward model and the discretization of the action space. This combination allows the controller to effectively search the tree of potential future states. Results show that it is indeed possible to implement a competent MCTS-based racing controller. The controller generalizes to most road tracks as long as a warm-up period is provided.

1. INTRODUCTION

Monte Carlo Tree Search (MCTS) is a best-first search method that builds a search tree iteratively, and makes decisions based on the statistical analysis of simulated play-outs [4]. The most popular version of MCTS, *Upper Confidence Bound on Trees* (UCBT), was first formulated in 2006 [11]. The algorithm has since received significant attention for its ability to make educated guesses in situations with search spaces too large for exhaustive search methods. MCTS received its initial breakthrough displaying significant potential in turn-based games such as Go [19] and Hex [1]. It has since been adapted to real-time applications as well, such as Ms. Pac-Man [16], Super Mario Bros [9] or StarCraft [6, 10]. These adaptations are non-trivial, as these games carry a very different set of challenges, and have necessitated the development of new modifications of the core MCTS algorithm.

Similarly, simulated car racing presents interesting challenges to artificial intelligence (AI) methods. Additionally, for computer controlled agents it is so far difficult to perform at a competitive level. The reasons for this are manifold. Car racing domains are characterized by a wide, continuous action space, real-time decision making, and the need for at least some level of foresight if optimal play is desired.

Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015), June 22-25, 2015, Pacific Grove, CA, USA.
ISBN 978-0-9913982-4-9. Copyright held by author(s).

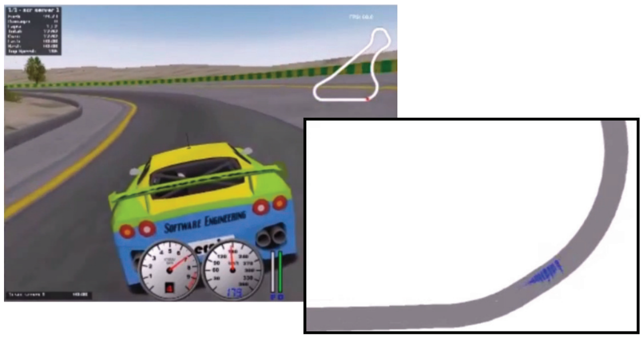


Figure 1: MCTS Car Controller in TORCS. The lower right window displays a graphical representation of the tree search, with blue dots representing tree nodes superimposed on a two-dimensional drawing of the track model. A video of the developed MCTS TORCS controller can be seen here: https://youtu.be/GbUMssvolvU.

As far as we are aware of, MCTS or similar tree search methods have not yet been applied to simulated car racing, which is not surprising, given how different car racing is from the sequential, discrete, zero-sum board games on which MCTS has shone in the past. On the other hand, testing the limits of a method and adapting it to a new problem, can bring important insights.

The Open Car Racing Simulator (TORCS) is a popular platform for experimenting with different AI methods in car racing. A variety of approaches have been developed [3, 5, 18, 8, 12] and a scientific competition (the “Simulated Car Racing Championship”) is held annually under IEEE auspices [14, 15]. This competition provides a set of rules that levels the playing field between potential solutions, as well as a programmatic client-server framework that enforces these rules.

Most prevalent in the competition are solutions involving artificial neural networks (ANNs), evolutionary algorithms or a combination of the two. Planning algorithms in general, but tree search methods in particular seem very under-represented, making it an ideal avenue to explore. Furthermore, ANN-based approaches often lack foresight, and creating a controller that generalizes across different tracks can be difficult. A planning-based method is a natural candidate for approaching the problem from a different angle.

Given the lack of research on applying planning algorithms to car racing, and drawing inspiration from the recent successes of MCTS, this paper presents a TORCS controller that makes its decisions exclusively based on tree search with simulated play-outs. The goal of this article lies in examining the feasibility of MCTS in TORCS. Our solution has been developed within the confines of the competition framework, partly for its convenience, but also for the sake of comparing our results with results from previous competitions as a general measure of utility.

2. RELATED WORK

Cardamone et al. [5] successfully applied a genetic algorithm to find the optimal race line in TORCS from internal track file data. In racing, a race line is a path along the track that a driver may follow to complete it. This is used to great success, beating the heuristic approach by “Simplix" [20], the best controller that was available for TORCS in 2010. However, in the TORCS competition, the controllers do not have direct access to the track files, thereby limiting the application of Cardamone et al.’s approach. Their work does however highlight the strength of a controller that is knowledgeable about the shape of the track ahead of it, allowing it to optimize the path it will follow.

Quadflieg et al. created a model of the track solely from sensor data [18], arguing that obtaining knowledge of the track can pave the way for a wider application of planning algorithms. The track construction algorithm in this paper takes inspiration from their approach, but to run the simulated MCTS play-outs, a more detailed model than their method can provide becomes necessary.

Research into modeling a forward model has been done by Butz et al. [3]. In their work, its application is directly in the sensor space and not combined with a track model, which limits its use in regards to planning, and in particular with regards to a tree search in the space of the car’s planar positioning. However, it does make for an interesting comparison between utilizing a learned forward model and using a physics-based abstraction of the internal engine, such as in this paper.

3. MCTS FOR SIMULATED CAR RACING

Monte Carlo Tree Search is a best-first tree search algorithm, in which the search space is probed by stochastic sampling through simulated play-outs. A node in the tree represents a state in the search space, and one new node is added to the tree at every iteration of the search. The search is *anytime* – it can be halted and polled for an answer after any number of iterations. One iteration involves four steps: tree policy, expansion, default policy and backpropagation [4]. MCTS relies on a forward model of the game mechanics to perform simulations. Sometimes the game itself provides this, in other cases, such as ours, the model is an external approximation (see section 3.1).

During the **tree policy** step, child j is selected to maximize the following formula:

$$UCB_j = \bar{X}_j + C_P * \sqrt{\frac{2 * ln(n)}{n_j}}, \tag{1}$$

where UCB_j is called the Upper Confidence Bound for node j, $\bar{X}_j$ is the average value of node j, n_j is the number of times node j has been visited, and n is the total number of visits to the parent node. The C_P constant determines the weighting between exploration (trying out a less-visited node) and exploitation (selecting a so far seemingly promising node).

The node found with the tree policy is **expanded** by applying the forward model one time-step. In the **default policy** step, a random play-out is performed from the newly expanded node. In classic MCTS, this simulation runs until a terminal state is reached. Our solution contains a configurable limit on the number of time-steps a simulation may run. Therefore the approach requires a heuristic for evaluating the desirability of a given state at any time-step, which is further detailed in Section 3.3. As MCTS is anytime, it can be stopped at any point during the search. The longer it gets to run, however, the more likely it is to converge on an optimal solution.

Applying MCTS to car racing, especially when part of the official racing competition, brings its own challenges. The *tick* or time step under the competition framework is a mere 20 milliseconds long [13]. This makes sure the car is updated with a relevant view of the game world fairly often, but it also puts a limit on the runtime of the decision-making algorithm. For this reason, the tree search controller in this paper is designed to cache its search between ticks, making its decisions within an extended time interval. The car controller records the current state at time t, and applies the forward model once to obtain a state at time $t + t_{step}$, where t_{step} is the configurable decision time step. While driving towards this state in-game by repeatedly applying the best decision from the previous search, the controller will utilize its time budget searching for a decision from $t + t_{step}$.

In addition, utilizing tree search is intended as a mechanism for planning ahead, so it is natural that the presented approach involves a longer period of time between the steps at which new decisions are considered. In other words, a good decision should not have to be reconsidered immediately.

It is also important to note that the MCTS controller has not been augmented with special case handling such as ABS (anti-lock braking system), ESP (electronic stability program) or off track recovery that could potentially increase its performance. This decision stems from the goal to first have the MCTS controller stand on its own in order to better assess its feasibility.

3.1 Simulation Framework

In order for MCTS to work, two things are needed: a **forward model** capable of predicting future game states based on what actions are taken, and an **evaluation function** to probe the desirability of a given game state. The TORCS competition framework only allows access to data that is relative to the car’s position on the racing track at any given time step, which is not sufficient for simulating future time steps that require static knowledge of the track’s shape. However, the rules of the competition [13] allow for a warm-up period before the actual race begins. The presented approach takes advantage of this warm-up period by driving the car slowly and steadily around the track, recording sensor data for the construction of a track model.

This gives rise to a “simulation framework” consisting of three components: the track model, the forward model and the evaluation function. The track model is mapped out in Euclidean space, allowing the forward model to be implemented with vector-based calculations based on the laws of classical physics. We found this approach to a forward model more tractable than one based directly on the sensor data. Thus, two transformations are needed:

- Transformation from observed states in the sensor space to states in Euclidean space, for use with the forward model during simulations
- Transformation from states in Euclidean space back to the sensor space at the end of simulations, for use with the evaluation function

The control flow for one iteration of the algorithm can be summarized as follows:

1. Observe a state s in the sensor space
2. Transform s to the equivalent state σ represented in Euclidean space

3. Perform Monte Carlo simulations on σ by repeatedly applying the forward model

4. Transform σ back to a simulated state s' represented in sensor space

5. Evaluate s' with the evaluation function

The forward model has been implemented by approximating the relationship between the two dimensions of our action space and the car's position and velocity in Euclidean space, i.e. between pressure on the gas pedal and acceleration, and between rotation of the steering wheel and angular velocity. These functions have been inferred through experimentation and regression analysis on recorded sets of data.

It is important to note that for the initial experiments in this paper, data has only been collected on tarmac (regular road) tracks, limiting the utility of our forward model on other track types, such as hill and dirt tracks. We also lacked the means to collect accurate deceleration data. How this may have affected the controller is discussed in Section 5.

3.2 Action Space

An action in TORCS allows the control of a number of factors, but we have limited these to acceleration, braking and steering. Transmission is controlled by a deterministic algorithm based on RPM, and acceleration/braking is further reduced to a single value *gas* for simplicity, with a negative sign representing brake.

Thus an action is defined as the combination of only two values: one representing gas or brake pressure, and one describing steering. Both are continuous values, and are discretized in the interest of having as small a branching factor as possible. Both gas pressure and steering are discretized into intervals of 0.2 and 0.1, respectively, and only selection of neighboring values to the previous action are allowed. For example, with a current gas pressure of 0.4 and an interval of 0.2, it will only be possible to take actions with a gas pressure of either 0.2, 0.4 or 0.6.

While this method limits the possibility to make sudden and drastic changes in action, the discretization allows for a finer action granularity, reduces skidding (which is currently not part of the forward model), while still maintaining a low branching factor. Braking and steering at the same time are also not allowed, as this combination almost always leads to skidding.

3.3 Evaluation Function

Equation 2 describes the formula for evaluating the desirability of a given state. The state space does not contain successful terminal states, since the benchmark is based on driving as far as possible within an unlimited number of laps. Unsuccessful terminal states (denoted by $\perp$) occur when the car is outside the marked boundaries of the track:

$$\begin{aligned} k &= \frac{d - d_0}{\frac{1}{2} a_{max} t^2 + v_0 t}, \\ k_\perp &= k^2 * P, \end{aligned} \tag{2}$$

where d is the distance along the track line in the given state, d_0 is the distance along the track in the state at the root of the search tree, v_0 is the velocity at the root, t is the time difference, k is the value in a non-terminal state and $k_\perp$ is the value in a terminal state. P is a configurable constant between 0 and 1 (the penalty factor). The denominator $\frac{1}{2} a_{max} t^2 + v_0 t$ describes the equation of motion.

Terminal states are penalized by being squared. Since values themselves fall between 0 and 1 (ideally), this penalizes high-valued terminal states less than low-valued ones. For configurability, this is further modified by the parameter P.

A heuristic that normalizes the distance traveled along the track line by the total length of the track is not ideal, since this yields values that grow increasingly larger as the game progresses. The C_P constant of the UCT formula in Equation 1 has to be configured based on the interval of the value range. Thus, the formula in Equation 2 yields a value relative to the *local* outset of the search. A value of zero means the car has been motionless for the full duration of the time difference t between the local outset and the current state. A value of one means the car has accelerated as much as possible, following the track line perfectly for the full duration. The intention is to have an admissible heuristic, i.e. bounded by the maximum acceleration a_{max}. However, since the distance is measured along the middle of the track, and not along the optimal race line [2], we cannot make this claim.

Table 1: Average Transformation Error

Sensor	Avg. discrepancy	CG Speedway
Distance from start	0.053 m	2057.56 m
Track position	0.027 m	15 m
Angle to track	$2.2 * 10^{-5}$ radians	N/A

Table 2: Simulation Framework Performance Impact

Measurement	No. of iterations	Fraction
Isolated tree search	1937.7	100%
Only transformation	1413.8	73%
Only forward step	1242.3	64%
Full framework	842.2	43%

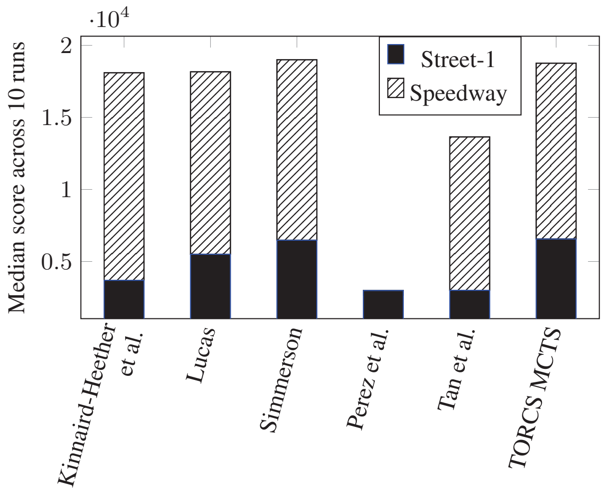
4. RESULTS

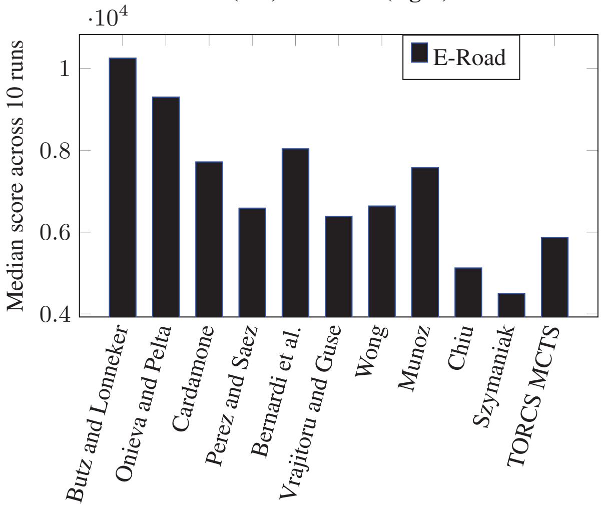
In order to properly review the performance of the MCTS car controller, it is important to first establish the impact of the simulation framework on the accuracy of the controller's decision-making. Table 1 shows that the transformations between states in the sensor model and vector model are off by a slight margin of error. The second column displays the average discrepancy between the input state and output state for the given sensor after two reflexive transformations, measured every tick for five seconds, totaling 250 measurements. The third column displays the total dimensions of the CG Speedway track for comparison. During tree search, states are not transformed back and forth multiple times, so a discrepancy of five centimeters is acceptable, and unlikely to be the cause of suboptimal benchmarks of the controller as a whole.

Table 2 gives an idea of how much the simulation framework impacts the runtime performance of the tree search algorithm. The number of iterations possible with a timeout of ten milliseconds on an isolated tree search with no simulation framework being invoked are compared to the same benchmark with transformations and/or forward stepping being applied in the search. Having the full framework present cuts the number of iterations to 43% compared to its absence.

Figure 2 shows how our own controller, labeled TORCS MCTS, compares to the results of the car racing competitions held in 2008 [14] and 2009 [15]. For the competition, a set of three tracks is se-

Figure 2: Car racing competition results on tarmac roads from 2008 (left) and 2009 (right).





lected and each controller is run ten times on each track for 10,000 game ticks. The median of the distance raced is then used as a score. A video of the MCTS controller in TORCS can be seen here: https://youtu.be/GbUMssvolvU.

In 2008, five controllers were submitted and trialed. Three of these, namely Tan et al., Lucas, and Kinnaird-Heether et al. , had major parts or even all of their controlling logic handwritten. Perez et al. used a genetic algorithm, and Simmerson used NEAT. In the 2009 competition, twelve controllers were submitted at various stages, of which the five best controllers all had handwritten parts, including one or more for crash recovery, ABS and ESP, which is currently not part of the MCTS controller (Section 3).

The three selected tracks in the 2008 competition consist of one hill track, one tarmac track, and one oval track. For the 2009 competition, the tracks consist of one hill track, one dirt track and one tarmac track. As our forward model is based on data from tarmac tracks, we are unable to compete on dirt and hill tracks; our scores were consistently lowest on these tracks. As there are no relevant comparisons to make beyond this fact, these results were omitted from Figure 2.

Compared to the 2008 controllers, the presented MCTS controller makes first place on Street-1 and hits the top three on Speedway. The 2009 controllers drive significantly better, and the MCTS controller only achieve scores comparable to the bottom three controllers. Comparisons were not made with more recent competitions due to their use of auto-generated tracks. The readily available standard tracks of previous competitions thus provided the only reliable track-by-track benchmark.

5. DISCUSSION

The developed MCTS controller handles tarmac roads well, and by fine-tuning the various parameters an increase in the controller's performance on the benchmarks was observed. The driving behavior shows promising tendencies, such as accelerating on straight sections and braking before turns.

The three tracks chosen for the competitions usually contain at least one track that is either a dirt track or a hill track. A dirt track has low traction, which causes high speed on acceleration to spin or skid the car. A hill track has steep climbs and descends, making the car accelerate slower or faster than on a flat track. Both these types change the basic premises for how the car behaves on the track. Since these variations are currently not part of the calculations in the forward model, the MCTS controller is unable to race these tracks at competitive speeds. In the future, changes to the forward model together with tests of traction during the warm-up period will likely significantly increase the performance on dirt and hill tracks. Collecting more accurate data on acceleration, deceleration and angular velocity, possibly in combination with a sophisticated learning algorithm rather than regression analysis, could further improve the performance of the controller.

As the number of actions was limited to discourage heavy braking and steering at the same time, bringing the car in a good position before a corner is crucial for competitive racing. If the car is on the wrong side of the track before a corner, it has to change positions before being able to hit the brake. This requirement can lead to a variety of unforeseeable outcomes, from not taking the corner optimally to not taking it at all. How the controller behaves on a very sharp hairpin (turn 11) in the provided Street 1 track, exemplifies this challenge. The speed of the car needs to be very low, but the turn ends the longest straight segment of the track. The time it takes to brake is therefore high, as the car enters the braking zone at high speed. If the car is not positioned well when the search reaches the upcoming turn, it needs to adjust its position to make a proper entry. The time it takes to make this adjustment prevents a successful deceleration in time, and the car either comes to a complete halt or leaves the road. We attribute this behavior to the relatively low depth of the search tree. With a greater depth, the controller might be able to predict that the turn is coming up sooner, thus repositioning the car before entering the brake zone.

6. CONCLUSION

This paper presented an MCTS-based controller for the TORCS racing game. The search is tuned to maximize the distance raced within the allotted time. To perform simulations, a track model was built and better driving capabilities were obtained by pruning the action space and removing actions with outcomes known to be undesirable. The MCTS controller handles tarmac tracks well, but fails when driving on dirt or hill tracks. This discrepancy is to be expected as the forward model is currently optimized for tarmac roads. Compared to the results from the 2008 and 2009 car racing competitions, an MCTS-based approach may present a viable option for controlling racing cars at competitive speeds.

7. REFERENCES

[1] B. Arneson, R. B. Hayward, and P. Henderson, “Monte-Carlo tree search in Hex", *IEEE Transactions on Computational Intelligence and Games*, 2010, pp. 251–258.

[2] B. Beckman, “The Physics of Racing, Part 5: Introduction to the Racing Line", 1991. http://www.miata.net/sport/Physics/05-Cornering.html

[3] M. Butz and T. Lönneker, “Optimized Sensory-motor Couplings plus Strategy Extensions for the TORCS Car Racing Challenge", *IEEE Symposium on Computational Intelligence and Games*, 2010, pp. 317–324.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods", *IEEE Transactions on Computational Intelligence and AI in Games*, 2012, pp. 1–49.

[5] L. Cardamone, D. Loiacono, P. Lanzi, and A. Bardelli, “Searching for the optimal racing line using genetic algorithms", *IEEE Symposium on Computational Intelligence and Games*, 2010, pp. 388–394.

[6] D. Churchill, and M. Buro, “Portfolio greedy search and simulation for large-scale combat in StarCraft”, *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013, pp. 1–8.

[7] K. Deb, A. Pratap, and S. Agarwal, “A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, 2002, pp. 182-197

[8] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, “Robust player imitation using multiobjective evolution", *IEEE Congress on Evolutionary Computation*, 2009, pp. 652–659.

[9] E. J. Jacobsen, R. Greve, and J. Togelius, “Monte Mario: Platforming with MCTS", *Proceedings of the 2014 conference on Genetic and evolutionary computation*, 2014, pp. 293–300.

[10] N. Justesen, B. Tillman, J. Togelius, S. Risi, “Script- and cluster-based UCT for StarCraft”, *2014 IEEE Conference on Computational Intelligence and Games (CIG)*, 2014, pp. 1–8.

[11] L. Kocsis, and S. Levente, “Bandit based monte-carlo planning", *Machine Learning: ECML*, 2006, pp. 282–293.

[12] J. Koutnik, G. Cuccu, J. Schmidhuber, and F. Gomez, “Evolving large-scale neural networks for vision-based reinforcement learning”, *Foundations of Digital Games*, 2013.

[13] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated Car Racing Championship Competition Software Manual", April 2013.

[14] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez, “The WCCI 2008 Simulated Car Racing Competition", *IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 119–126.

[15] D. Loiacono, P. Lanzi, J. Togelius, E. Onieva, D. Pelta, M. Butz, T. Lönneker, L. Cardamone, D. Perez, Y. Sáez, M. Preuss, and J. Quadflieg, “The 2009 Simulated Car Racing Championship", *IEEE Transactions on Computational Intelligence and AI in Games*, 2009, pp. 131–147.

[16] T. Pepels and M. H. M. Winands, “Enhancements for Monte-Carlo Tree Search in Ms Pac-Man", *IEEE Conference on Computational Intelligence and Games*, 2012, pp. 265–272.

[17] D. Perez, E. J. Powley, D Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, “Solving the physical traveling salesman problem: Tree search and macro actions”, *IEEE Transactions on Computational Intelligence and AI in Games*, 2014, pp. 31–45.

[18] J. Quadflieg, M. Preuss, O. Kramer, and G. Rudolph, “Learning the track and planning ahead in a car racing controller", *IEEE Symposium on Computational Intelligence and Games*, 2010, pp. 395–402.

[19] A. Rimmel, F. Teytaud, L. Chang-Shing, Y. Shi-Jim, W. Mei-Hui, and T. Shang-Rong, “Current Frontiers in Computer Go", *IEEE Transactions on Computational Intelligence and AI in Games*, 2010, pp. 229–238.

[20] “TORCS Robot Development | Simplix". http://www.simplix.wdbee-aorp.de/SIMPLIX/SimplixDefault.aspx