38

An Architecture Overview for AI in Racing Games

Simon Tomlinson and Nic Melder

38.1 Introduction

This article describes the requirements, architecture, and best practices for high-speed vehicle racing AI. We mainly consider high production value simulation games, but arcade style games can also be built using simplified elements of the approach here. Subsequent articles in the Racing section of this book will then expand in greater detail some of the critical aspects of the racing AI system.

Many of these techniques have been known for some time [Biasillo 02a, 02b, 02c] but in racing, the difference between winning and losing can be small fractions of a meter or second. Because of this it is important to push for accuracy and detail with the objective of convincing the user that his AI opponents are as expert on the track as he is.

38.2 Understanding the Physics

Before designing the AI system, it is important that the developer fully understands the type of physics in play and how that will affect the implementation. In very simple games the AI will advance along a predefined spline, and all that is needed is emulation of cornering speed, acceleration, and braking based on parametric formulas and basic collision detection—examining if the spline ahead is blocked within the braking distance.

There is no physics required. But in a full simulation, the AI car will be similar to that experienced by the player, including an engine model, transmission, tire models, braking model, and all the vehicle quirks and characteristics this encompasses.

Vehicle physics is mainly about forces and tires. Any acceleration on the car, either a speed change or a change of direction, results in forces being applied through the tires. The magnitude of the forces is limited by the capability of the tire and, if that limit is exceeded, the tire will no longer grip the road surface and the car will slide. In corners this can result in oversteer (the car digs in towards the apex and the rear slides out) or understeer (the car drifts to the outside of the bend). When accelerating or braking, the tires might spin or lock, reducing the effectiveness of the speed change, so a key action of the AI is to ensure that the available tire grip is not exceeded. This is done by ensuring corners are taken at the correct speed, braking is done in good time, and acceleration is managed with gradual pressure on the throttle. If the AI is too conservative it will be slow, so the best AI, like the best human drivers, will need to work very near "the limit of grip" at all times. This is what the whole AI design is geared towards.

The details of racing physics are beyond the scope of this article, but it is highly recommended that the AI developer gains a solid grounding in this area [Pacejka 06].

38.3 The Architecture

It is best to consider the AI in terms of four layers and one subsystem that extends through these layers. The top layer is the *character layer* or *persona layer.* This layer works on a fairly long timescale and is responsible for the individual driver's skill levels which affect the performance of the AI. The next layer is the *strategic layer,* which works on short timescales between a single frame and a few seconds. The strategic layer houses the behavioral elements of the system and determines broad steering and speed goals based on an examination of the track representation in the near and medium distance. The next layer down is the *tactical layer,* which will normally be processed every frame and is responsible for refining the steering and speed goals into solid values. Typically, there will be a number of competing goals produced at the strategic and tactical level, and these will be analyzed and combined in the final *control layer.* This layer is responsible for calculating the controller inputs (steering, brake, and throttle) in order to achieve the previous layers' strategy.

Alongside these layers is the vitally important *collision avoidance subsystem*. This can usually be split into two sections based on the immediacy of the analysis they perform. At a longer range, the AI will need to look at the track some distance ahead of the car and plan safe routes around any obstacles, such as debris or damaged vehicles. At a shorter range, the AI will need to make small but rapid adjustments due to static or dynamic obstacles. Examples include racing close to a solid wall, alongside another car when overtaking, or driving close behind an opponent to 'slipstream' in preparation for an overtake.

38.4 The AI Driver Persona

In real life no two drivers will be the same, they will have different abilities and approach racing with slightly different strategies. In a fixed formula race, such as Formula 1 or IndyCar, where the performance of the vehicles is quite similar, differences in driving skill will substantially contribute to the race result. At a minimal level, a spread of driver skill will produce a spread of the physical positions of the cars on the track. This is not just more realistic, but also helps to ensure the player experiences a number of micro challenges as they progress through the field. Done well, driver personas can feed into the observed behaviors and add significant color to the game experience. However, such color is only really worthwhile if the actions of the AI driver are distinguishable in the game. It also helps to support these traits with driver profiles viewable pre-game or perhaps in-game signals like audio commentary.

The primary characteristic will be *skill*, a measure of their ability to drive at the limit; secondary characteristics may include *aggression*, *vehicle control*, and *mistakes*. Primarily, the skill characteristic would be applied where any speed related calculations are used, such as multiplying the actual available grip in calculations of cornering speed and braking distances. This will have the net effect of reducing the overall speed. While this characteristic might be exposed to the user and designer in the range [0, 100], the requirement that we always drive close to the limit means that, within the AI strategic layers and below, this should be mapped onto a smaller range of say 98–99%. Although this may appear as a small range, a 1% variation accumulated over several laps can have a surprisingly large effect, especially when applied in several places in the code. It may be found that a much smaller range in the skill factor is necessary to avoid the AI spreading unrealistically. Similarly, by mapping to a lower range (e.g., 80%–82%) the overall race experience should become easier so the skill characteristic can also be used to modify the overall game difficulty.

Using a smaller range makes it more difficult for the user to discern the speed variance between the different drivers. To counter this, a *biorhythm* can be applied to the driver's skill. The skill factor will vary slowly with time, for example using a sine wave with a period of 100 seconds. In this way, the driver's average skill measured over a number of laps might be 99%, but for some short periods of time, the skill factor could be significantly lower. This will make the AI driver temporarily vulnerable to being overtaken without having an overly large long term effect. It is not necessary to use a sine wave as the biorhythm, and it is worth experimenting with different waveforms, for example, a square wave with a low duty cycle that produces a skill factor near the upper limit for 90% of the time but perhaps a skill as low as 90% or 95% for short periods. With a defined set of rules, this idea can be further extended to encompass a full race-pace system [Jimenez 09, Melder 13].

Secondary characteristics can be used to affect behavioral changes such as the probability of entering overtaking mode, the rate that the throttle is applied, or the number of unforced mistakes that they make. A driver with a high *aggression* characteristic will drive closer to the cars in front and will require less space to overtake a vehicle, a driver with a low *control* characteristic may take twice as long to push the throttle in, and a driver with a high *mistake* characteristic might lead to random errors in corner speed calculation.

38.5 Racing Behaviors

The breadth of behaviors in a racing AI system is not large, so generally a finite-state machine (FSM) is sufficient to represent them. At any time most of the behaviors may become valid and so they usually all compete to be the active one, and should be reviewed on every update of the strategic layer. A good way to manage this is for each valid state (as defined by the current state exit transitions) to evaluate a 'utility' score. Provided that this utility score is larger than the current state's, a state transition would result. In order to avoid short-lived states and rapid transitioning, a small additional threshold should be added to the current state utility. This will produce hysteresis which will tend to retain the current behavior state unless a significantly stronger alternative arises. Some of the more common behaviors are described below.

38.5.1 Normal Driving

The objective of this behavior is to simply get around the main track as fast as possible. The AI should stay reasonably close to the optimum racing line, but remain conservative in terms of avoiding other cars: allow a reasonable amount of space between any car in front or that happens to be alongside. The utility for this behavior is a baseline constant for the whole system that other behaviors must beat. If using a utility range of [0, 1000] a value of 500 is appropriate.

38.5.2 Overtake

In *overtake* mode, the AI is actively seeking a line which will allow it to pass one or more cars in front. The AI is still basically driving along the track, but this is modified in order to achieve an overtaking maneuver. Dealing with more than one overtaking target simultaneously can be necessary in situations such as the race start, as otherwise the resultant behavior may look uncompetitive and mechanical. The analysis for the state is to look ahead down the track. If there are one or more opponents in range and the AI has a speed advantage, the analysis will determine a safe (wide enough) window across the track width for overtaking.

The deeper this analysis, the better overtaking events will be and anticipation is key. For example, the car ahead may not be slow enough *now*, but if it is approaching a corner it is likely to slow down and offer an opportunity. Another example is the car in front is under-steering and will shortly open up a viable gap on the inside.

Not every overtaking opportunity should be taken and randomness or a biorhythm trait should contribute to the utility calculation. Generally, overtaking is less likely on a straight where all cars are at full throttle, so the current and future track features should figure in the utility. If there is a significant speed advantage (such as an opponent slowing due to a mechanical failure), then overtaking should always be activated. Note that this behavior should not go into too fine a grain in terms of dimensional precision, since when the AI does overtake, the tactical layer and short range avoidance subsystems will deal with the detail of driving past the opponent(s) safely.

Once overtaking mode is activated, the driver should become a little more aggressive and even closer to the limit. Not withstanding any mechanical boost, such as using nitro, the AI skill levels should be increased close to saturation or indeed beyond. For example a driver might deliberately choose to accept the risk of exceeding their grip limits temporarily and skidding slightly (under-steering) by braking late to get past the opponent as they enter the bend on the inside, in the knowledge that once they have claimed the position and blocked the other car they can then regain composure and complete the turn. Avoidance tolerances should also be reduced; the AI will be prepared to get closer behind or alongside an opponent while overtaking. Indeed in some game types some light, controlled contact might be allowed.

The strategic analysis of the overtaking opportunity should continue so that, for example, if the gap closes, the overtake is aborted and the AI returns to normal driving mode. To avoid erratic AI switching, it is also good to disallow re-activation of the overtaking state for a few seconds after an aborted attempt. A successful overtake is where the AI has passed the opponent and is some distance ahead, in which case it can return to normal driving. The AI should not switch too early, though, as a small reduction in speed due to the state change could mean the positional advantage is quickly lost as the opponent comes back past the AI.

38.5.3 Defend and Block

In some racing styles it is acceptable to try to defend an overtake by moving across into the path of the opponent. This state would be triggered using a utility where an opponent was approaching from behind, not directly on the same line and with some current or future potential speed advantage.

The *defend and block* state operates by making small steering moves to match the AI's line on the track with the opposing vehicle behind. Depending on the racing rules in force, defending may be aborted once the opponent gets alongside or even after a set duration or lateral movement has been exceeded (Formula 1 rules for example allow a single lateral move, i.e. you cannot weave back and forth).

In all but the most aggressive racing genres, defending should be terminated if you are putting the opponent in danger—for example, pushing him off the track. A good way to manage this is to allow the opposing AI or human to post a *complaint message* if they feel they are in danger, so that the defending mode is aborted. Similar to overtaking, while the AI is performing a defend and block, the speed margins should be pushed to the limit.

38.5.4 Branch

Some racing genres use tracks with multiple routes and branches; even in classic simulation games, there may be a pit lane. The utility value to decide to take a branch could be based on all sorts of strategic considerations: tire wear, fuel levels, and position relative to other cars in a simulation genre, or an analysis of the value of a short cut against the risk in a more arcade style game. In either case, the key detail is to make the decision early enough so that the correct line into the branch can be followed. Also, the AI needs to be extra aware of nearby vehicles that might block access to the branch—if necessary, braking to fall behind the other vehicle in plenty of time. For that reason, this behavior should not end until well into the alternative track or, for example, the branch is aborted due to obstructive vehicles.

38.5.5 Recover

In the most realistic games, the AI can make occasional mistakes resulting in going off road or spinning within the track bounds. Thus, the objective of a recovery behavior is to get back racing. In practice, on- and off-track recovery may be separate behaviors triggered using metrics, such as pointing the wrong way along the track or being too far out of the main track bounds. In either case, the first task is to stabilize the car, stop sliding, and slow down or even stop if facing the wrong way or close to a barrier.

In off-track recovery, the goal is to drive towards the nearest edge of the track at a modest speed, giving way to any cars that are approaching from behind. For on-track recovery, the AI can do a slow tight turn or may even try to spin up the back wheels for a *donut*. In either case, good driving manners mean that, until the upstream road is clear, the AI should just stay put to avoid causing a collision.

Because the recovery process is very disadvantageous in terms of competitiveness, the AI should be allowed every chance to carry on racing where possible, so it is best not to trigger recovery based on a single update of the utility value, but rather to integrate up the utility over a few seconds. For example, at a strategic update rate of 10 frames per second and a utility range of [0, 1000], one might add 10 points for every meter off the track and 10 points for every wheel that has lost grip, but also subtract 20 points every frame even if off-track. Once the car is on-track, stable, and facing the correct way, the points are quickly subtracted, leading to the completion of this behavior.

38.6 The Race Choreographer

The race choreographer is essentially a scripting system that can be used by designers to affect storyline aspects within the course of a race. It is a separate AI object, receiving event triggers from the individual AI vehicles, the physics system, or the main game code. It then interacts with the AI at the persona or strategic level (although it could also implement actions on the physics or other systems). Examples of events include changing a specific driver's skill level halfway through the race, causing a tire blowout, or triggering an AI custom behavior (e.g., try to collide with the player).

38.7 Interfaces

The main interface between the AI and the vehicle should be the same as that between the human controller and the car, so that the AI and a human are interchangeable. The AI will also need an interface to extract metrics from the physics, such as available grip and whether each tire is currently sliding. And there is no harm in one AI vehicle asking another for detailed information such as, "Are you about to turn left?" This may seem like cheating, but a real life experienced driver will always be able to read the signs and interpret an opponent's intentions. Of course, a real driver may not get it right all the time, so adding some fuzziness and randomization within the AI request interface might be appropriate.

38.8 Balancing the AI

The AI must always be balanced and tuned to make sure each car can drive as quickly as possible, but also to maintain a good distribution of vehicles through the track so that the player always feels "involved" with the other cars. Balancing the AI can often be the most difficult part of the game development process. It can be very time consuming and stressful, so it is always worth considering how this process will work and what tools can be built to ease the pain.

The objective is to maximize the experience for the player; we want them to have to work hard to compete and overtake, but ultimately we want players of varying abilities to still have a reasonable likelihood of winning the race. In a scenario with tightly restricted car specifications, like Formula 1, the problem is mainly to bring a group of quite similar

AI opponents' performance into proximity with the player. In a game with a wider range of vehicle performance on the same track, the problem is even greater.

It is for this reason that many games employ a system called *rubber-banding* [Melder 13] which attempts to change the AI speed to best match the player's over the course of the race. Normally, the objective is to be beating the player at the start of the race, but to be losing at the finish, ideally such that the player reaches a winning position in the last few hundred meters of the race. Note that the mechanics of balancing may not lie solely within the AI. If the AI is pushed past the 100% grip limit it will make mistakes, slide, and ultimately its performance will suffer, thus hindering the balancing process.

It is usually better to consider the AI balancing as a process of achieving the correct spread of performances over the group, while the overall balance against the player is better achieved using adjustments within the vehicle physics such as tire grip, engine power, and torque. Also bear in mind that in some implementations, the AI uses a "cut-down" version of the physics, which may not produce the same outcomes as the human's vehicle, even where all the parameters are equal. This must also be accounted for with balancing adjustments in the physics.

38.8.1 Offline Automated Learning

Even where a dynamic in-game system is not used, there is substantial opportunity for optimizing the AI performance using offline automated learning. Initially during development, the definition of the racing line is usually specified by the designer placing splines directly on the track, but this can then be improved. One simple improvement is to use bisection to optimize each node on a racing line one by one [Biasillo 02c]. However, those who are familiar with multivariate optimization know that this sort of strategy can be prone to false solutions.

Broadly, the issue is that a racing line is a sum of its parts, or rather each part or node is not independent of those around it. Consider a single node on a curve containing 10 nodes which is smooth but not optimal. If a single node is moved 0.5m laterally, all this will do is produce a kink in the racing line that is likely to have the effect of slowing down the AI. In practice, what needs to happen is that several points should be moved coherently and in proportion. A good technique to achieve this is a modified genetic algorithm. This is based on the random Monte Carlo technique where a number of variables are randomly selected and randomly modified and, if the measured process metric (usually the average lap time) improves, the new solution is kept, otherwise it is discarded, and a new attempt is tried.

In a multivariate problem, Monte Carlo is generally of the order of $\sqrt{N}$ faster than a linear search, where N is the number of variables. Where genetic algorithms improve efficiency is that they deal with groups of variables in the problem together, patching lines of several nodes together from two parents. If this is combined with coherent mutation (modifying a run of adjacent nodes), then this automated technique can be improved greatly. Moreover, it can be very advantageous to use knowledge of the track in the construction of the GA. For example, marking up groups of nodes that are known to form a curve and encouraging mutations that follow a pattern which is likely to be favorable will help the GA find a good solution. Note also that optimization of the racing line is not the only area for automated learning. Parameters used with the strategic layer, such as utility weights and the constants within the control layer or PID controller, are also ripe for this technique.

There are a number of pitfalls and tips when using automated learning. As with any numerical optimization method, the difficulty increases geometrically with the number of parameters being optimized in the problem. Thus, if automated learning is to be used, the AI should be designed from the start in a way which minimizes the amount of data. For example, in defining the racing line, nodes should be placed wider apart to minimize their number around the track.

Another pitfall with automated learning is that it can be difficult to see how the solution has arisen, and any changes to the underlying system that require a re-evaluation of the solution can be time consuming. For this reason, a manual editing method should always be provided as a fallback. However, other optimization methods can also suffer similar issues. For example, in a regime where the racing line is simply recorded using a human player, a substantial change in the track will still require the whole track to be re-learned, recorded, and tested.

Ultimately, though, the final arbiter of whether a balancing process has worked or not must be a human. In such a complex process as a race, even though AI lap times say they are optimized, it may be that the AI is not competitive in real terms. This is perhaps the case because the AI runs slower when other cars are on the track or perhaps because the AI are too easy/hard to overtake. Testing with a range of player abilities over varying scenarios will always remain an essential balancing tool.

38.9 Conclusion

This article has described in detail a broad architecture for the AI in a high-speed racing game. The detail and depth of the implementation will depend on the style of racing game, but most games will require most, if not all, of the elements described. However, in real life, racing can often depend on small fractional details, and the same is true in a high-end racing simulation. In that respect we have only really provided a starting point; there is much more to learn and many areas to experiment with in the quest for realism, performance, and excellence.

A racing AI implementation is full of paradoxes. The principle paradox is that the AI must operate in real-time, making complex decisions in order to not only remain in control of the car, but to drive it at the very edge of its capabilities. However, the success of the AI lies in long hard hours preparing prebaked data, tweaking parameters, and anticipating solutions to potentially difficult scenarios. Thus, the paradox is that a successful racing AI is the careful balance of real-time control and carefully thought-out data. Moreover, you should not skimp on the design phase; in a highly connected AI system the discovery of a weakness in the latter stages of development can be very difficult to remedy.

References

[Biasillo 02a] G. Biasillo. "Representing a race track for AI." In *AI Game Programming Wisdom*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2002, pp. 439–443.

[Biasillo 02b] G. Biasillo. "Racing AI logic." In *AI Game Programming Wisdom*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2002, pp. 444–454.

[Biasillo 02c] G. Biasillo. "Training an AI to race." In *AI Game Programming Wisdom*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2002. pp. 455–459.

[Jimenez 09] E. Jimenez. "The Pure Advantage: Advanced Racing Game AI." http://www.gamasutra.com/view/feature/3920/the_pure_advantage_advanced_.php, 2009.

[Melder 13] N. Melder. "A rubber-banding system for gameplay race management." In *Game AI Pro*, edited by Steve Rabin. Boca Raton, FL: CRC Press, 2013.

[Pacejka 06] H. B. Pacejka. *Tyre and Vehicle Dynamics,* 2nd edition. Oxford: Butterworth-Heinemann, 2006.