CS 1130A

Artificial Intelligence for Games

David Beirne

0705140

Racing Game AI

An Investigation into AI Techniques for Motorsport Simulation Games

14/12/2007

Contents

1 Abstract

As the Senior System Analyst of a fictitious Computer Games Development Company, this paper aims to investigate and critically evaluate both existing and possible upcoming approaches to creating believable Artificial Intelligence driven opponents and spectators found in a typical sports car racing videogame.

David Beirne
BSc(Hons) Computer Games Programming
University of Abertay Dundee

2 Introduction

This paper takes key elements of the motorsport simulation genre and identifies AI techniques that can and have been used in commercial applications in solving the specified problem. The paper discusses possible solutions to the problems of track representation, finding racing lines and the use of a driver model for controlling AI based opponents. By citing proven successful approaches from top-selling applications in the genre and detailing the AI techniques which make them successful. The paper also proposes some new applications of AI in racing games, specifically in an AI-based car modification and tuning application, as well as an investigation into how existing pedestrian and spectator models can be improved with the use of AI.

3 Track Representation

Since the majority of racing occurs on a track, one of the most important elements in creating a working racing game is the design of the data structure to represents the track.

3.1 Possible Solutions

3.1.1 Minimalistic Approach

The minimalist approach consists of a linked list of nodes which form a track. Each node contains its position and a pointer to the next node, along with an acceleration value. Each car starts at a starting node, from there they accelerate at the speed specified in the node towards the position of the next node as specified.

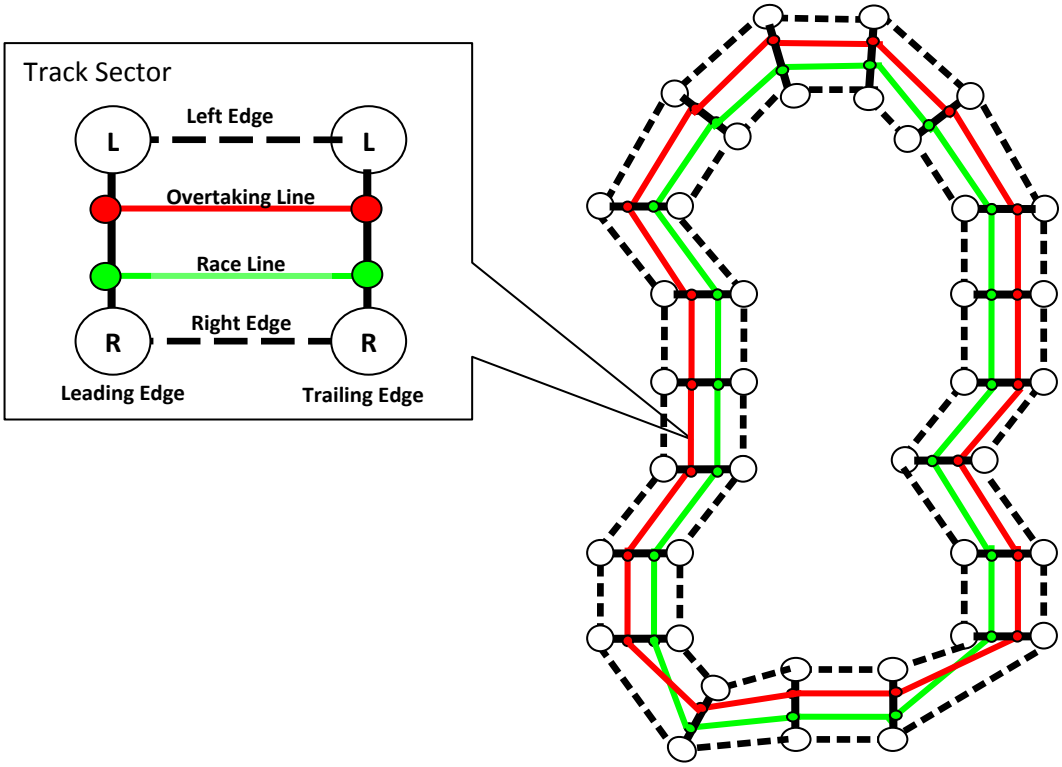
This approach could be combined with a finite state machine within the car to handle avoiding other cars and/or overtaking as well as a recovery state if the car gets stuck, would result in a very simple racing game AI. Such an approach would be almost trivial to implement but would not be incredibly realistic.

3.1.2 Node List Approach

Gari Biasillo of Electronic Arts Canada provides a series of articles on track representation and racing AI logic published in “AI Game Programming Wisdom” by Charles River Media. Biasillo concisely describes a track represented by a double-linked list of sectors which are described by interfaces which in tern identify the racing track where trailing and leading interfaces are shared to reduce memory usage. Each sector of the track consists of 4 planes, which describe the bounds of the track at the current sector (where each sector must be convex), a driving line node and an overtaking line node.

Each line node contains its starting world position, its lengths, and a forward and right direction vector. The forward vector is found by subtracting the sectors start position and end position (start position + length) and then normalising it after zeroing its Y component. The right vector

is perpendicular to the forward vector and is used to describe how far the AI car is from the racing line. The reason for zeroing the Y component of the forward and right vector is to simplify calculating the cars orientation relative to the XZ plane. By also projecting the cars forward vector onto the XZ plane, the problem becomes 2D.



Aside from a geometric representation of the track, Biasillo also proposes that further data is provided to the AI by each sector. These include a "Path Type" value which describes the path ahead, allowing the AI to make decisions in the event of multiple racing lines, a "Terrain Type" which would allow a car to determine whether it was capable of traversing the next sectors terrain (if for example it was damaged or the terrain was too rugged). "Walls" which allow the AI to know if it should keep to one edge so as not to be hit into walls by opponents. "Hairpin Turn" which allows the AI driven car to shorten its viewing vector so that it doesn't try to take a shortcut across a hairpin (due to the opposite side being closer to the next edge as the crow flies). Finally a "Break/Throttle" value which could be used to specify the state of the cars breaks from -1.0 (full throttle) to 1.0 (full breaks). These flags, or similar, would also be useful in a rally game where the co-driver must yell out instructions to the player on the upcoming corners and dangers. [Gari Biasillo - EA. (2002) , AI Game Programming Wisdom, Charles River Media]

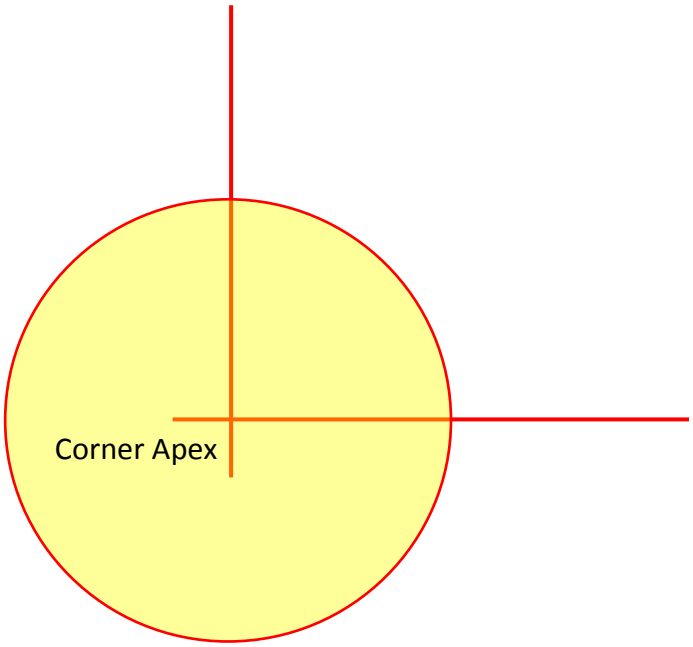
Biasillo's suggested track representation seems like a very clean and simple approach to what seems at first to be a complicated problem. The proposed representation could be easily implemented and adopted. Further extensions of the interfaces could add multiple racing lines and, as Biasillo suggests in his article, alternate objectives aside from taking the fastest line,

such as collecting weapons and health, or considering more diverse terrains and car damage levels.

4 Racing Lines

The term racing line is used to describe the most efficient path a vehicle can take through a single corner, a set of corners or the entire lap of a track. As the shortest distance between two points is a straight line, a racing line on a corner will generally be characterised as being the largest turn radius possible within the boundaries of the track, creating the most direct rout possible.

A simple example would be a constant-radius right turn. In most cases (when factors such as bumps or variations in banking or traction at different places in the turn are negligible), the ideal line would start on the far left hand side of the track upon entering the corner and to exit the corner at the furthest left point:



[Prima Press. (2003) ,Forza Motrosports Guide (edited), http://assets.xbox.com/en-us/PrimaGuides/Forza.pdf]

The two points of entrance and exit are connected by an arc which runs smoothly from start to finish with the largest available radius (to maximize cornering speed). In the majority of cases, this involves the arc smoothly intersecting the apex of the corner.

The Analysis of a racing line containing multiple straights and corners is somewhat complicated. The net goal of the driver is to minimize the time spent traversing the course, this may sometimes involve deviating from the typical racing line to set up properly for the next corner or straight. Exit speed onto a straight is considered to be one of the most important aspects in racing, as an exit speed only slightly lower corresponds to a substantially larger time traversing the straight (the speed reduction is

"integrated" across the straight). In this case, the driver might aim for a later apex and thus straighten the car out earlier. This will allow maximum throttle to be applied earlier and increase the speed out of the bend.

4.1 Possible Solutions

4.1.1 Fixed Racing Lines

Due to the complexity of this problem, the majority of racing game AI systems rely on a pre-defined racing line (or set of racing lines) defined by the games developers during track creation or generated interactively as the result (or average result) of extensive play tests and test drives where an expert players laps are recorded.

4.1.2 Dynamic Racing Lines

Another approach is not to use fixed racing lines at all, but have the AI driving model capable of finding its own racing line during the race as will be discussed later in the Driving Model section.

5 Driving Model

The most common way to represent an AI controlled car is in exactly the same way as the car is represented to the player. That is, the AI is able to control a car by inputting numerical values to the its control interface. Generally a player will have an analogue trigger or button for accelerating which transmits values within the range of either -1.0 (reverse) to 1.0 full acceleration or 0.0 (stopped) to 1.0 full throttle - depending on whether a “gear” is used for reverse or not. The player also has a break which ranges from 0.0 to 1.0 from off to fully engaged. Steering is represented by a -1.0 to 1.0 range from fully left to fully right and any direction between. Changing a gear is normally achieved by a shift-up and shift-down button or automatically determined by the game (depending on the players choice).

The aim of the driving model is to convert information about the cars current speed, position and distance from the racing line to input analogous to a users input. The advantage of this is that the game would then need only one “Car” class which can be controlled by either AI or a player through a common interface. This is also important for players, who do not want to feel that their AI driven opponent has an unfair advantage over them.

“Like most driving games…it was a case of getting the AI to follow the racing line. This is where the driving model comes in. The model takes information about the car’s state e.g. position, speed; and the racing line in the vicinity. Using that information, it calculates controls for the car i.e. left/right/accelerate/brake. This I’m sure is typical of driving games.” [Jeff Hannan. (~2000) ,AI Junkie Interview, http://www.ai-junkie.com/misc/hannan/hannan.html]

5.1 Possible Solutions

5.1.1 Artificial Neural Networks

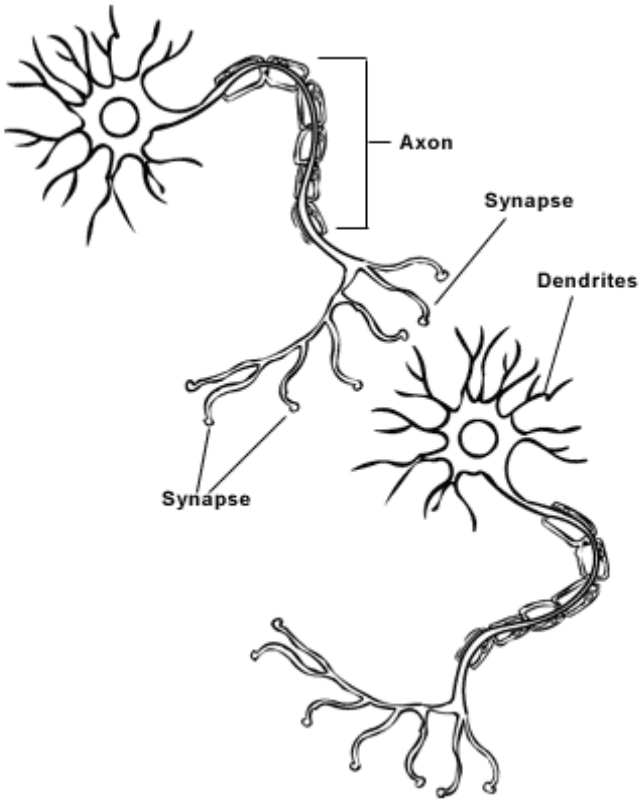
Hannan's Driver Model, which was used in Colin McRae Rally 2.0, was based on his previous PhD research on Neural Networks "The approach that I used I developed while doing my PhD. Its common sense really, in that by experimenting as much as possible, you get a greater idea of what is working and what isn't. I think in the early days of Neural Nets, when processing time was more expensive, there was perhaps more of a tendency to try and get everything right with one shot, because that one shot would take a week. I basically started at the most simple structure, and added more neurons until performance increased no further." [Jeff Hannan CodeMasters. (~2000) ,AI Junkie Interview, http://www.ai-junkie.com/misc/hannan/hannan.html]

5.1.1.1 What is an ANN?

An Artificial Neural Network (ANN) is a network of individual neurons (known as Perceptrons) which can be used when there are a large number of possible conditions and inputs or where a finite state machine would be too complicated. ANNs are capable of learning, and using previous experience to improver performance. They can be used for recognizing patterns and generalizing output from a small set of training data.

5.1.1.1.1 The Perceptron

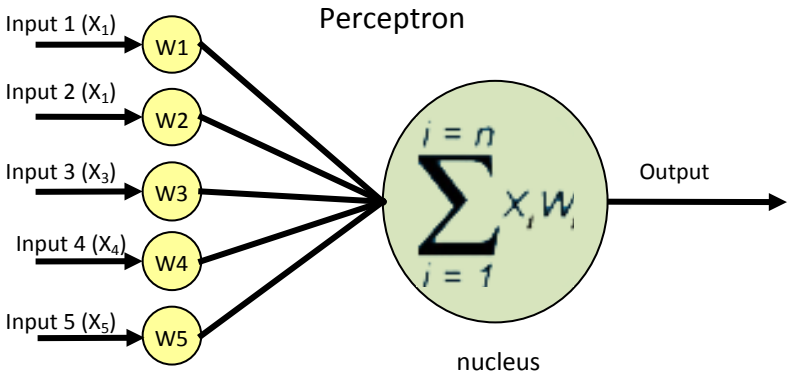
Much like the brain cell it is modelled on, the perceptron has a number of input channels and a single output channel (analogous to the dendrites and axons of a biological neuron).



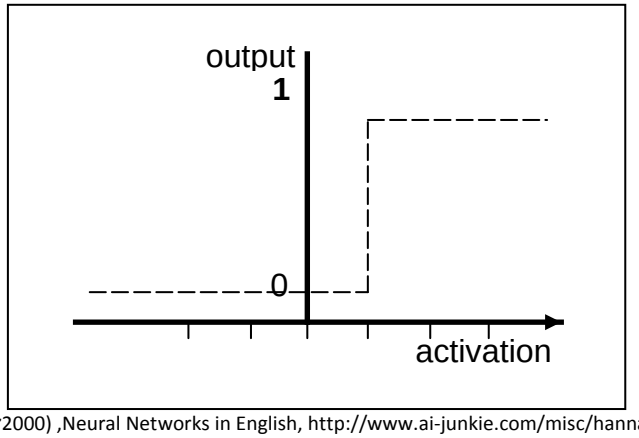
[NIDA. (2006) ,Neuron Image, http://www.nida.nih.gov/JSP/MOD3/page3.html]

"The interface through which they interact with surrounding neurons usually consists of several dendrites (input connections), which are connected via synapses to other neurons, and one axon (output connection). If the sum of the input signals surpasses a certain threshold, the neuron sends an action potential (AP) at the axon hillock and transmits this electrical signal along the axon."[Wikipedia. (Unknown) ,Biological Neural Networks,http://en.wikipedia.org/wiki/Biological_neural_networks]

perceptions work in much the same way. The perceptron has a number of weighted input channels and a single output value. If the sum of the inputs multiplied by the weights given to those inputs reaches a threshold value, an output signal is sent to the next perceptron.



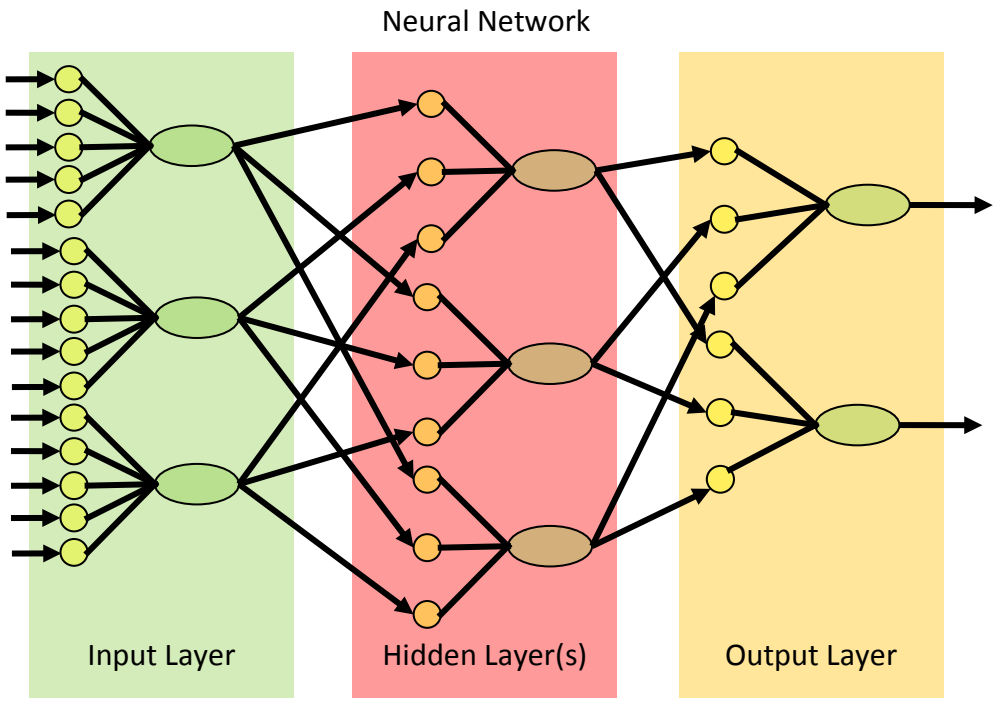
"The weights in most neural networks can be both negative and positive, therefore providing excitory or inhibitory influences to each input. As each input enters the nucleus it's multiplied by its weight. The nucleus then sums all these new input values which gives us the activation (again a floating point number which can be negative or positive). If the activation is greater than a threshold value - lets use the number 1 as an example - the neuron outputs a signal. If the activation is less than 1 the neuron outputs zero. This is typically called a step function:



[Mat Buckland. (~2000) ,Neural Networks in English, http://www.ai-junkie.com/misc/hannan/hannan.html]

5.1.1.1.2 Feed Forward Network

The feed forward network is created by linking a number of perceptions together where the output of each neuron is fed forward into the input of the next:



A neural network usually consists of an input layer and an output layer usually connected by one or more hidden layers . There can be any number of neurons in the network and any number of hidden layers, usually these are the minimum number required to solve the problem. Hannan describes his approach to creating a neural network based driver model as “experimental...I basically started at the most simple structure, and added more neurons until performance increased no further”. [Jeff Hannan, CodeMasters. (~2000) ,AI Junkie Interview, http://www.ai-junkie.com/misc/hannan/hannan.html]

5.1.1.1.2.1 Input Layer

For a racing game the inputs would be values such as the cars current distance from the racing line, its current acceleration, its current breaking value, the position of the player, the roughness of the ground below it, the amount of damage to the cars chasse etc. “I experimented with every input I could think of and calculate. Adding an extra input parameter and running the process described above should give better performance, provided that input contains some relevant information that helps the network classify the outputs. However, its ideal to use as few inputs as necessary... I can’t say exactly which inputs I used, as that’s commercially sensitive

information you know. But, the particular ones I settled on were really down to the nature of the physics model in the game. They will probably be different anyway for any other driving game or simulation." [Jeff Hannan, CodeMasters. (~2000) ,AI Junkie Interview, http://www.ai-junkie.com/misc/hannan/hannan.html]

5.1.1.1.2.2 Output Layer

The outputs from the neural network would be the resulting action taken by the driver model. In this case, the controller values such as steer -0.4, accelerate 0.9 break 0.1 etc. The difficulty comes in finding the layer or layers between the input and output which decide how the output is reached.

5.1.1.1.2.3 Hidden Layer(s)

Each input is sent to every neuron in the hidden layer and then each hidden layer's neuron's output is connected to every neuron in the next layer. There can be any number of hidden layers within a feedforward network but one is usually enough to suffice for most problems you will tackle.

5.1.1.1.3 ANN Training

Once the neural network has been created it needs to be trained. One way of doing this is initialize the neural net with random weights and then feed it a series of inputs which represent, all possible configurations. For each configuration we check to see what its output is and adjust the weights accordingly so that whenever it sees something looking like a "right turn" it outputs a 1 and for everything else it outputs a zero. This type of training is called supervised learning and the data we feed it is called a training set. There are many different ways of adjusting the weights, the most common for this type of problem is called 'back propagation'.

Hannan makes clear that the use of neural networks should be limited to a specific problem. "The neural nets are constructed with the simple aim of keeping the car to the racing line. They are effectively performing that skill. I felt that higher level functions like overtaking or recovering from crashes should be separated from this core activity. In fact, I was able to work out fairly simple rules to perform these tasks." [Jeff Hannan, CodeMasters. (~2000) ,Generation5 Interview, http://www.generation5.org/content/2001/hannan.asp]

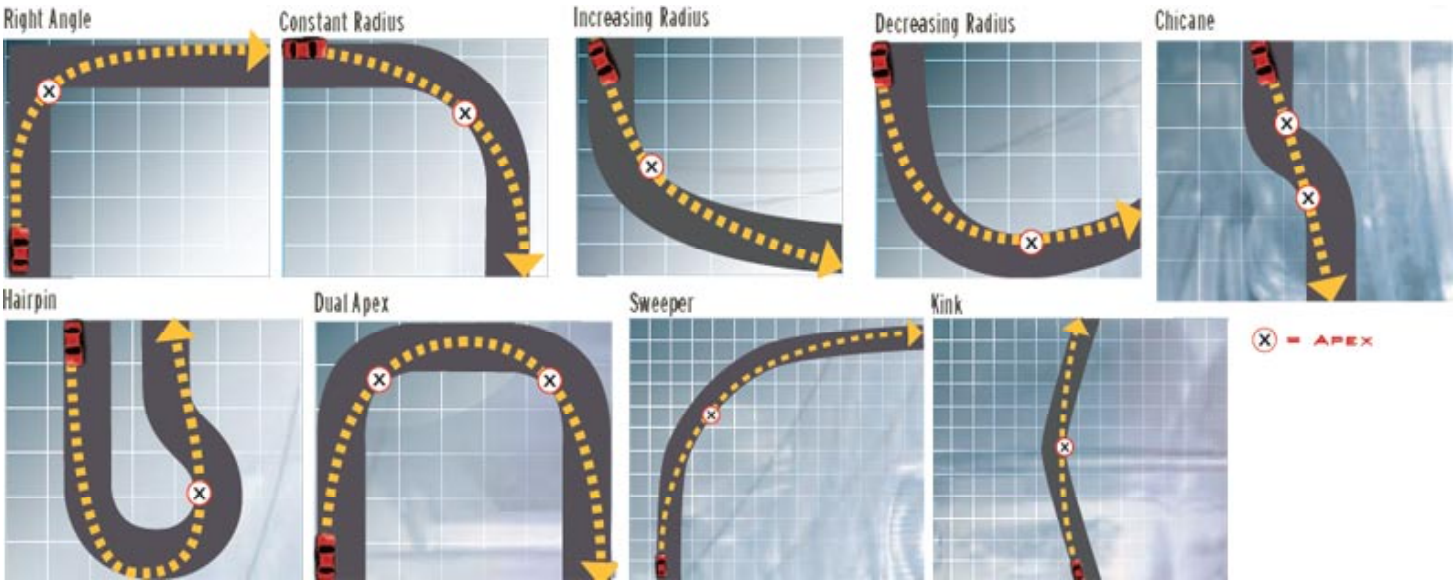
5.1.2 'Machine Learning'

One existing racing game; "Forza Motorsports" created by Microsoft Game Studios for the Xbox (and its sequel on Xbox360) does no use fixed racing lines. "The AI in Forza doesn't have fixed lines to follow. Whilst we can't really give away how it works at the moment, suffice to say that you'll see the Forza AI following a variety of lines during races. They dynamically change their racing line based on opportunity and conditions. At the easiest levels, you'll see a fair amount of variety and different approaches to corners. As you progress through the higher difficulty levels you'll notice your opponents being more consistent in their lines and showing less

deviation. So, pretty much like humans of different abilities drive" [Microsoft Research. (2005) ,Forza Motorsport AI Site, http://research.microsoft.com/mlp/forza/]

Forza prides itself on its machine learning implementation called Drivatar (Driving Avatar). Drivatar was developed by the Machine Learning and Perception group at the Cambridge Microsoft Research Institution and is utilized in two areas of the game: As a "learning" game feature where players can create their own AI driver, and as the underlying model for all the AI competitors in the "Arcade" and "Career" modes.

Players are encouraged to consider the Drivatar to be a clone of himself, which can be employed to complete some of the games career mode challenges if the player is having problems with a particular course or just wants to sit back and watch the race instead of competing in it. A Drivatar training section of the game, allows the player to teach their Drivatar to play like them by undertaking five lessons. "The Lessons are devilishly cunning because they shoe-horn you into teaching your Drivatar how you drive a representative sample of corner types that appear throughout the game...once you've completed all five lessons, you'll have taught your Drivatar how you drive each and every corner type in the game" [Microsoft Research. (2005) ,Forza Motorsport AI Site, http://research.microsoft.com/mlp/forza/]. The Forza manual specifies the nine corner types found in the game:



Completing the five lessons gives the Drivatar enough information on the players driving style to create a model based on the following characteristics:

- **Variety** – how consistent the player is.
- **Line** – how smoothly the player guides the car around the corners and through combinations thereof.
- **Entry Speeds and breaking points** – how early does the player break before entering a corner and whether they are conservative or reckless.

- **Apex Speeds and positions** – how close the player is to the apex of the corner and how fast they are going.
- **Exit Speeds and acceleration points** – when the player starts accelerating as they leave a corner and how well they have maintained their speed around the corner.

Once the Drivatar has this information, it is considered "mature" and becomes available in the "Career" and "head to head" mode; where the player can race against their own Drivatar. One important point is that a Drivatar is never considered complete, and repetition of the training is encouraged. For this to work, the Drivatar stores a set of scores for each corner type, representing the average score the player has gained on each corner type where "percentage points are awarded for entry speed, exit speed, closeness to apex, smoothness of line and time taken through the corner" [Microsoft Research. (2005) ,Forza Motorsport AI Site, http://research.microsoft.com/mlp/forza/] .

The Drivatar based AI opponent cars are similar to the Drivatar trained by the player but are trained and stored prior to the game being published. The team behind Forza seem less willing to talk in detail about how their machine learning works for their AI opponent cars (for obvious reasons), but is possible to speculate that it starts out with a fixed set of performance scores and then uses a form of supervised learning, using the corner models as the test data to improve during races.

6 AI Car Tuning

Many racing games offer car tuning options from changing tires based on weather conditions to modifying air-fuel ratios and stiffening shock absorbers and sway bars. Usually these options are offered only to the player and not to the AI cars – who's attributes are normally predetermined.

6.1 Possible Solutions

6.1.1 Genetic Algorithms

Genetic Algorithms can be seen as an optimisation search algorithm which uses successive generations of agents, based on biological organisms, to find the most favourable solution from a great number of possibilities. Due to the relatively limited number of possible tuning options provided by most racing games, it is possible to propose the use of a Genetic Algorithm to find the best possible tuning options for a race on a given track in given conditions. In fact a similar approach has been taken by the Digital Biology Group who created a GA based "evolving car" which gained first place in a simulation of the Silverstone racing track (created by Electronic Arts). When pitted against an F1 car tuning expert and a car tuned by a research team, even managing to beat the real-life track record on the Silverstone track by almost a second. [Wired. (2004) ,Breeding Race Cars to Win, http://www.wired.com/cars/energy/news/2004/06/63900]

6.1.1.1 Biological Basis

Stored in the nucleus of every cell of every organism is a set of rules which describe how it is built. The rules are encoded in genes; genes are in turn strung together into chromosomes. Each gene represents a specific trait of the organism, such as eye colour and has several different settings i.e. blue, brown or green. The complete set of genes (the genome) is an organism's genotype, and the physical expression of the genotype – the organism itself is known as the phenotype.

Natural selection is described as the differential survival and reproduction of organisms with genetic characteristics that enable them to better utilize environmental resources. That is, the organisms which are better suited to their environment often have a better rate of survival and are therefore more likely to reproduce.

The sexual reproduction of two organisms results in individuals who share half of their two parents' genes. This process of sharing is called cross-over or recombination. Sometimes a gene is mutated at this stage resulting in a completely new trait but this is rare.

Genetic Algorithms are a way of solving problems by mimicking these natural processes. Genetic Algorithms use the same combination of selection, recombination and mutation to evolve a solution to a problem.

6.1.1.2 Implementation

The first step in implementing a Genetic Algorithm is deciding which traits should be stored in the organisms' genome and which data structure should be used to store them. Common data structures are binary strings, arrays of strings and trees. When searching for an optimal car configuration the genome should contain every possible modification value.

Next a suitable test is determined to see which organisms are "fitter" than others. In a racing game, a suitable test would be to have each organism race around the same racing line with the same acceleration values. The games physics model would determine which organisms get to the finish line first, as the only thing effecting performance would be the cars tuning values.

Finally the "fitter" cars are selected to pass their genes onto the next generation. Those that reach the finish line first should be selected and recombined to make new offspring. After a number of generations, the best suited car should be found.

Expert Systems

An expert system is software that simulates experts in a field and uses their knowledge to provide solutions to problems posed by the user.

A common implementation of an expert system is a computer program, which when posed questions by the user, generates one or a number of solutions or conclusions. The system emulates reasoning capability to reach its solutions through mathematical analysis of the problem input by the user.

In motorsport simulation games, an expert system could be used to select the correct tyres for the weather conditions for example. The expert system could take into account the current weather (clear, light rainfall, heavy rainfall or snow for example) and select the most suitable available tyres, for example road, off-road, snow-tyres, off-road snow chains). This decision from the expert system could also take into account the percentage of road to mud on the track or even the frost on the road. The expert system would work as a pit-man to select the best possible tyres for the next race based on a simple or complex number of known inputs.

There are two main ways an expert system could come to the conclusion for the tyre example above. These are Forward chaining searches and Backward chaining searches.

6.1.1.3 Forward Chaining

Imagine the system knew the next race would be under snowy conditions where 85% of the track was road and 15% was mud. The system would search the if-clauses for example:

1. IF the weather is snowing THEN use snow tyres
2. IF the track is greater than 50% road THEN use road tyres
3. IF the track is less than 50% road THEN use off-road tyres.
4. IF the track is greater than 50% road and the weather is snowing THEN use road tyres.

etc.
the system would use these if-clauses to conclude the THEN clause as its result. in this case, number 4, because the track is greater than 50% road and the weather is snowing. The expert system would choose the snow-tyres. This is also known as the Data Driven method

6.1.1.4 Backward Chaining

Backward Chaining works back from the THEN-clause to reach its expert conclusion. This could be used to select the correct car suspension:

1. IF the weather is sunny THEN the track is hard.
2. IF the track is hard THEN use soft suspension.

Suppose a goal is to conclude to select soft suspension. The rule-base would be searched and rule (2) would be selected because its conclusion (the THEN clause) matches the goal. It is not known that the weather is sunny, so this IF statement is added to the goal list. The rule-base again searches the Then-clauses and this time rule (1) is selected because

its Then-clause matches the new goal just added to the list. This time, the if-clause (the weather is sunny) is known to be true and the goal to use soft suspension is concluded. Because the list of goals determines which rules are selected and used, this method is called goal driven.

7 Pedestrians & Spectators

Racing and driving games have often used pedestrians and spectators to make their track or level design more dynamic and realistic. While some games encourage interaction between players and the spectators / pedestrians, others keep a physical barrier between the two. As pedestrians and spectators are most often present as visual decoration and not integral to gameplay, it stands to reason that the majority of racing games do not invest the computational cost of running a complex AI system for each member, but rely on tricks and simple techniques – some of which are detailed in this section.

7.1 Possible Solutions

7.1.1 Fixed Animation

The simplest way to deal with the movement of Non-playable pedestrian characters is to assign them a fixed animation. Some racing games such as Project Gotham Racing 3 keep all spectators behind barriers so that the player never interacts directly with them. Spectators can be observed to repeat a small set of fixed animations including waving, jumping up and down and taking photographs.

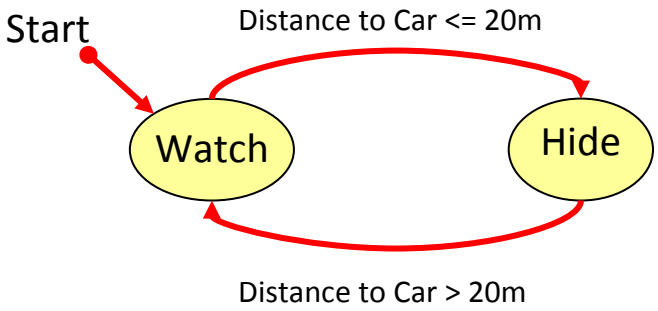
One of the positive aspects of this approach is that it not highly CPU intensive. This allows the Project Gotham 3 game to express huge numbers of fully 3d spectators in static positions.

[Project Gotham Racing 2. (2005) ,Crowd Image, http://pictures.xbox-scene.com/4/pgr3/hugecrowd_full.jpg]

However one thing that stands out in the game is that when the players' car slams into the barrier close to the spectators they seem not to notice and do not react. This is where some AI would be useful.

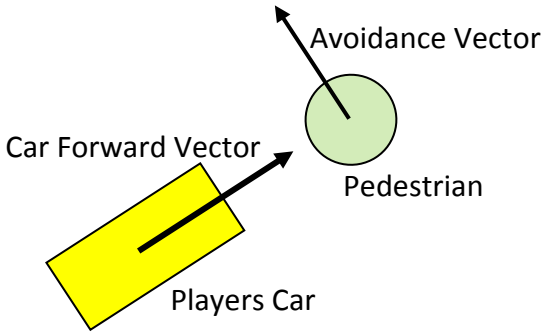
7.1.2 Finite State Machines

If each member of the Project Gotham crowd or even just the members towards the front of the crowd had a two node finite state machine, they could react accordingly if a car came within a certain distance of them or hit the barrier.



Without even changing their position, the crowd members could perhaps play a cowering animation, cover their face, or look away in shock. This would give the illusion that the spectators are actually watching what is happening and reacting to what they see.

Some driving games offer a more persistent world than Project Gotham Racing, where each pedestrian is apparently going about their daily business within a city. In Midtown Madness and Crazy Taxi, for example, there is no physical barrier between the players' car and the pedestrians, yet it is almost impossible for a player to run down pedestrians. Crazy taxi uses the approach that if the players car is within the bounds of a pedestrian, the pedestrian jumps away at a an angle perpendicular to the players forward vector.



Although this approach sometimes causes the pedestrian to jump through a nearby wall, it is very computationally inexpensive and effective in the majority of cases and could be extended in situations where more complexity is required.

In more violent games such as Carmageddon and Grand Theft Auto 3, the player is encouraged to run down pedestrians, in this case a combination of finite state machines and flocking can be used to create the illusion of a living city. "Grand Theft Auto has quite a range of pedestrian types, all of which are running different AI, based on function. In most games, this type of

behaviour is state based, probably with some messaging." [Schwab, Brian. (2004) ,AI Game Engine Programming, Hingham,MA, US: Charles River Media p.117]

7.1.3 Flocking

"Other systems use very simple flocking-type behaviours, with areas in the level being assigned particular values of attract and repel (thus, certain storefronts might attract people, who would look in the window for a while and then walk toward the

next attractor, whereas a dead body might be a powerful repelling force, so that people look like they're avoiding the accident). State of Emergency made good use of a system similar to this. The crowds were very fluid and reacted well to most of the action." [Schwab, Brian. (2004) ,AI Game Engine Programming, Hingham,MA, US: Charles River Media p.117]

7.1.4 Scripting

Some racing and driving games such as GTA involve a simplified RPG element, where non-playable characters can be spoken to in order to gain missions, work, or other objectives. These are normally pre-scripted events. Scripted events allow the games designers to

8 Conclusion

This paper has looked at five key elements of a racing game that may be either implemented with or improved with the use of artificial intelligence. The first section looked at how a track could be represented by either a linked list of nodes or a more complex system of interfaces and sectors constructed of edges to represent a more complete view of a track. The second section looked at the problem of defining or finding a racing line.

The third section looked at how two commercial games used AI techniques to control opponent cars around the track or along the racing line – it showed how Artificial Neural Networks could be used to drive a car with an interface analogous to that used by the player.

The fourth section looked at a new application of AI in racing games, proposing a car tuning system based on either genetic algorithms or expert systems.

Finally the paper looked at how AI can be used for pedestrians and spectators watching the races. This section proposed a system of finite state machines for controlling large numbers of spectators and considered other systems for more complex behaviours such as flocking and scripting.

While this paper has covered a lot of ground In AI techniques for racing games, there are still many more opportunities and uses for AI in motorsport simulation games .

9 References

1. [Gari Biasillo - EA. (2002) , AI Game Programming Wisdom, Charles River Media]
2. [Prima Press. (2003) ,Forza Motorsports Guide (edited), http://assets.xbox.com/en-us/PrimaGuides/Forza.pdf]
3. [Jeff Hannan, CodeMasters. (~2000) ,AI Junkie Interview, http://www.ai-junkie.com/misc/hannan/hannan.html]
4. [Mat Buckland. (~2000) ,Neural Networks in English, http://www.ai-junkie.com/misc/hannan/hannan.html]
5. [Jeff Hannan, CodeMasters. (~2000) ,Generation5 Interview, http://www.generation5.org/content/2001/hannan.asp]
6. [Wikipedia. (Unknown) ,Biological Neural Networks,http://en.wikipedia.org/wiki/Biological_neural_networks]
7. [NIDA. (2006) ,Neuron Image, http://www.nida.nih.gov/JSP/MOD3/page3.html]
8. [Microsoft Research. (2005) ,Forza Motorsport AI Site, http://research.microsoft.com/mlp/forza/]
9. [Wired. (2004) ,Breeding Race Cars to Win, http://www.wired.com/cars/energy/news/2004/06/63900]
10. [Schwab, Brian. (2004) ,AI Game Engine Programming, Hingham,MA, US: Charles River Media p.117]