Making Racing Fun Through Player Modeling and Track Evolution

Julian Togelius, Renzo De Nardi, and Simon M. Lucas

Department of Computer Science
University of Essex
Colchester CO43SQ, United Kingdom
{jtogel, rdenar, sml}@essex.ac.uk

Abstract. This paper addresses the problem of automatically constructing tracks tailor-made to maximize the enjoyment of individual players in a simple car racing game. To this end, some approaches to player modeling are investigated, and a method of using evolutionary algorithms to construct racing tracks is presented. A simple player-dependent metric of entertainment is proposed and used as the fitness function when evolving tracks. We conclude that accurate player modeling poses some significant challenges, but track evolution works well given the right track representation.

1 Introduction

The video game genre of racing games is almost as old as video games, and shows no signs of losing popularity; with every generation of new gaming hardware new racing games come out, with more complex physics, graphics and network connectivity. Whether any real progress is being made in racing game AI or track design is another matter. We have previously studied the application of evolutionary robotics techniques to developing controllers for cars in a simple racing game [1][2][3]. We have two distinct motivations: to create more adaptable, believable and entertaining games through using AI, and to explore the suitability of video games as environments for studying the emergence of general intelligence. In this paper, we explore the use of the framework developed in our previous work to create racing tracks tailor-made to maximize the enjoyment of individual human players.

While the use of logic-based AI in games has a history as long as artificial intelligence itself, and computational intelligence in games is now an established field[4], the use of AI techniques to enhance player satisfaction is a relatively new endeavour[5]. As far as we know, no-one has yet attempted this for racing games.

This paper is organized as follows: first we comment on the computational car racing problem in general and our racing game in particular. We then present the problem of optimizing racing game entertainment, and present our approach to this: player modeling and track evolution. Next, we briefly describe the car

controllers and evolutionary algorithm used. The following section is on player modeling, for which three different approaches are compared. Once a player is modeled, we want to evolve entertaining tracks for this player, so the final section presents our representation for evolvable racing tracks, our fitness functions and some results. We conclude with an outlook on possible future work.

With this paper, we are merely dipping our toes in a new and fascinating area, and both ideas and results are to be considered as preliminary. Due to space concerns, some technical details have been omitted, but these can be found in our earlier papers on evolutionary car racing.

2 The car racing problem

The focus of the earlier papers has been the following. We have one or several tracks, containing free space, non-permeable walls, starting positions of the car, and a chain of waypoints which the car must pass sufficiently closely in the correct sequence. We also have either one or two cars, each equipped with a sensor model that permits it to sense the immediate environment from its own perspective, and a discrete set of motor and steering commands. The physics of the simulation are reasonably realistic, accounting for inertia, drift, and semi-elastic collision between cars, and between cars and walls. The objective of the game is for the car to progress as far as possible along the track in 700 timesteps, which corresponds to 35 seconds when run in real-time. Progress is judged by the number of waypoints passed; for most tracks it is possible to complete several laps within the allotted time. At every timestep, the task of the controller is to choose, on the basis of available sensor data, one of three possible motor commands (forward, neutral or backward) and one of three possible steering commands (left, center or right). Given the momentum of the car and its turning radius, the action taken at any single timestep contributes little to the overall course of the car. The non-holonomic nature of the car and the solidity of the walls also make the car racing problem considerably harder than many simple mobile robotics problems, such as wall-following with a typical differential-drive robot.

In previous work, we compared a number of different controller architectures and sensor representations for this problem, and concluded that the configuration described below is the best performer overall[1]. In our experiments, the best evolved controllers outperformed all human drivers on the tracks tested. We also proposed methods for scaling up to several tracks[2] and cars[3] simultaneously.

3 What makes racing fun?

We have been unable to find any prior research on what makes it fun to play a particular racing game, or to drive on a particular track in that game. The initial hypotheses offered here are therefore based on our own experiences and on opinions gathered from an unstructured selection of non-experts.

– The sensation of speed is clearly a factor - people like to drive fast. A track should allow a high maximum speed.
– Driving on an endless straight track is not fun, even when driving at high speed. Tracks should be challenging to drive.
– Crashing all the time is not fun either. Tracks should have *the right amount of challenge.*
– People also tend to like a variety of challenges, so tracks should vary in character and not repeat the same kind of challenge all the time.
– Drifting or skidding in turns seems to be a major fun factor.

The renowned game designer Raph Koster, writing about video games in general, offers a different perspective[6]. According to Koster, playing and learning are intimately connected, and a fun game is one where the player is continually and successfully learning. One way to interpret this in the context of car racing would be that a good racing track is one on which the player does pretty poorly the first time he plays, but quickly and reliably improves in subsequent races.

4 Technicalities: sensors, neural networks and evolutionary algorithms

4.1 Sensors

For ease of comparison we use the same sensor setup as in earlier papers. All the control methods used in this paper use information from eight simulated sensors: the speed of the car, the angle to the next waypoint, and six wall sensors. Each wall sensor is located at the center of the car, and has two parameters under evolutionary control: its direction (expressed as the angle it makes with the axis of the car) and its maximum range, which may be anything between 0 and 200 pixels. A wall sensor returns a value between 0 and 1, depending on whether it detects a wall within its maximum range, and on the distance of the wall as a proportion of the maximum range.

4.2 Neural networks

Most of the car control methods are based on neural networks. These networks are all standard fully connected feedforward nets (MLPs) with the *tanh* transfer function. Only the weights of the networks are changed by evolution or backpropagation, the topology remaining fixed. Nine inputs (sensors and a bias input), six interneurons, and two outputs are used. The first output is interpreted as driving command (less than -0.3 means backward, more than 0.3 means forward and in between means neutral) and the second as steering command.

4.3 Evolutionary algorithm

Variations on an evolutionary algorithm we call *Cascading Elitism* (inspired by [7]) is used to evolve both tracks and controllers, problems which have several conflicting fitness measures. The algorithm is essentially an evolution strategy without self-adaptation, but with a simple modification in order to handle

multivariate problems. At the start of the first generation, a population of 100 genomes is created. In each generation, all genomes are evaluated according to the first fitness criterion, and the 50 best individuals (the first elite) are retained while the rest are thrown away. If there is a second fitness measure, the first elite are then sorted according to the new fitness measure and the best 30 individuals (the second elite) are retained; the same principle is used for the third fitness measure where the 15 best individuals of the second elite are kept. Mutated copies of the surviving individuals are then used to fill the population up to the initial level of 100. To eliminate the risk of disruptive effects such as ''competing conventions", crossover is not used. We have not studied the effect of varying certain parameters, such as population size, selection pressure, and order of the fitness functions. When evolving a car controller, the mutation operator applies perturbations drawn from a gaussian distribution with standard deviation 0.1 to all weights and parameters. For track mutation methods, see section 6.

5 Player modeling

In order to use evolution to automatically create an entertaining track for a particular human player, we need to be able to test the generated tracks against the relevant human player thousands of times. Subjecting any real human to such treatment would surely remove any entertainment value, so we need a way of reproducing the player's driving style on tracks on which he has not yet driven.

Charles and Black[8] survey player modeling in general, and Yannakakis and Maragoudakis[9] use player modeling for optimising satisfaction in a version of the Pac-Man predator-prey game. Much closer to our problem domain, the popular racing game *Forza Motorsport* for the XBox allows its players to train a "drivatar" to drive just like them. This is accomplished by dividing all tracks into segments, and recording the precise path the player takes through each segment[10]. When the drivatar subsequently drives a track it has not been trained on, this track is analyzed into segments and a new path is calculated that the car has to adhere to. Crash recovery is a hard-coded mechanism. While the *Forza* approach has proved very successful, it imposes severe constraints on the track designs, and requires information about the preferred path to be available to the controller, something we wish to avoid.

Other relevant prior research include the ALVINN experiments, in which a real car learned to drive on real roads using backpropagation, visual inputs and human driving, although it should be noted that the objective was to learn driving in general rather than a particular driving style[11].

5.1 Learning behaviour

Backpropagation Our first attempt at player modeling was very straightforward: learning a car controller from a human example. A human player drove a number of laps around a track, while the inputs from sensors and actions taken by the human were logged at each timestep. This log was then used to train a

neural network controller to associate sensor inputs with actions using a standard backpropagation algorithm. Several variations on this idea (such as more sensors, larger networks, and selective editing of the log file before training) were tried with very little success. Although the training often achieved low error rates (typically 0.05), none of the trained networks managed to complete even half a lap. Usually the car just drove straight into a wall, though in some cases it did not move at all.

Nearest-neighbour classification We also tried controlling the car with a simple nearest-neghbour classifier. In these experiments, we used the same training data as in the experiments above, and let the controller choose the outputs of the training data point that closest matched the current sensor inputs. The small amount of noise that is applied to sensors guarantees that the car does not simply replay the human action. The resulting behaviour looked very "human-like", indeed very like the driving style of the particular human being modeled. It was quite good as well, sometimes taking a lap or two at decent speed. But almost invariably it ended up taking a turn just a bit too early or too late and therefore crashed into a wall. After that, it was unable to back away from the wall. Another problem with this control method was that it had serious problems with generalization.

5.2 Learning characteristics of behaviour

Given the limited success of learning player behaviour directly, we decided to try an indirect approach, namely identifying measurable characteristics of the human player's behaviour, and adapting an evolved controller to reproduce these characteristics. We decided to use total progress along the track, the variance of progress along the track between different trials, and the number of action changes (changing from one driving command to another), as these characteristics are easily quantifiable and vary considerably with driving styles. In the evolutionary runs, we started from good pre-evolved controllers, which the multi-stage evolutionary algorithm further evolve based on logs of human driving.

Results were mixed. While it was not very hard to adapt the controller to exhibit the same progress as the human on the same tracks, the other characteristics were harder to learn. It was possible to evolve a low variance in progress, but only occasionally did we manage to evolve a higher variance. Trying to adapt the controller to produce a given number of action changes was hopeless. One of the authors changed driving command 69 times in 700 timesteps, while an evolved network made about 250 changes, a difference which could not be evolved away.

5.3 Learning performance profiles

We then reasoned that if we could not capture the actual driving style of a human player with the methods at our disposal, we could at least capture the unique performance profile of that driving style. For example, assume player

A drives rather recklessly, and so does well on straights and broad curves but often crashes into walls in narrow passages and turns, and player B drives very carefully and so maneuvers through narrow passages well but doesn't exploit his vehicle's full potential in straights. We would then want the model of player A to make better progress than B on tracks rich in straights and broad curves, and the model of B to do better than A on tracks requiring lots of precise maneuvering, even if the micro-features of the models' driving look nothing like those of the corresponding human players. To accomplish this, we designed three test tracks

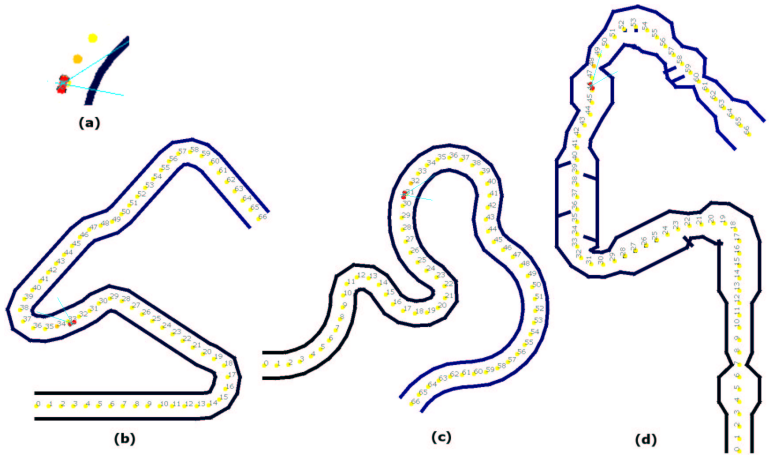

Fig. 1. Handcrafted test tracks

(see figure 1) to emphasize different aspects of racing skill. The first track consists of continuous gentle curves, the second track has long straights and sharp turns, and the third track has lots of narrow passages and obstacles. The progress of two of the authors on these tracks within 700 timesteps was measured (0.94, 0.86, 0.81, and 1.49, 1.49, 1.46 laps respectively). Controllers that approximated these performances were found within 100 generations.

5.4 A non-trivial problem

We see our attempts at player modeling as having been a partial success. On the one hand we have been able to produce controllers that closely match the performance profile of human players, but on the other hand we failed to model human driving behaviour more directly. The controllers modelled on performance profiles differ from humans in their micro-behaviour both qualitatively, in that their driving looks a bit artificial, and quantitatively, e.g. when counting action changes. We believe that these problems are due both to the reactive nature of all the controllers examined so far, and to the limited sensory inputs; the

sensor data we humans use to decide on the car's movement are much richer, and our complex neural networks allow such strategies as the construction and exploitation of forward models of the car. Much work remains to be done here.

6 Track evolution

In this section we define concrete fitness measures based on the considerations in section 3, and propose an evolutionary approach for track design. Many different representations of the racing track are possible; here, we propose one that emphasizes smoothness of the fitness landscapes. The track is encoded as a fixed length sequence of segments, all with the same midline length (the length of the track is therefore fixed). Each of the segments can represent a short part of a straight path, or of a curve (three different curvature radii are possible). In order to allow for more flexible tracks, each segment can have its own final width (three different widths are possible), subject to the constraint that the initial width has to match the width of the previous segment. Segments containing obstacles (in the center or in one side of the track) were also available as part of the genetic material. The mutation operator used by the evolutionary algorithm works simply by changing a segment into one of a different type with uniform probability 0.1. It is clear that the fitness obtained by a car driving on the mutated track will be similar to its fitness on the original track since, from the car's perspective, a large part of the track has not changed at all. This concept directly translates into a smooth fitness space by minimizing disruptive mutations. The chosen representation theoretically still allows for the track to turn back on itself and overlap, but no attempt was made to prevent this; in the unlikely event of this happening, the fitness obtained by the track would be very low and so the track would not survive the selection process. To ensure a set of viable individuals in the starting population, the initial elite consists of tracks using only straight segments. The tracks produced by this method will not in general form closed circuits, and so we artificially close the tracks by means of 'teleportation'. As soon as the car reaches the end of a track it is moved to the start, keeping the same position, velocity, and orientation relative to the track.

Fitness measures Three different fitnesses have been chosen to drive the evolution process: target final fitness, maximum speed, and maximum fitness variance. The first and most important fitness used, is given by $f_1 = 1 - \mid p_f - p_t \mid$, where p_f is the final progress (fractional number of laps) and p_t is the desired target progress. In order to smooth the effect of the sensor noise, the progress fitness of the single track is averaged over 10 repetitions. The final progress is intimately related to the average speed achieved during the test, therefore setting the target final progress corresponds to setting the type of track (slow or fast) that we are trying to produce. The second fitness measure is the maximum speed achieved within the trial; the aim of this fitness is to force the inclusion in the track of at least one section on which the player could possibly reach a very high speed. To drive the search towards challenging tracks, the third fitness selects for tracks

with a high variance in the final progress. Other fitness measures considered include speed variance and number of turns taken.

Results In general terms the track evolution process was considered successful; the evolutionary algorithm is indeed able to produce a track on which the player-modeled controller achieves the desired progress fitness. Changing the final progress target allows the production of a low speed or high speed track. We have already commented on the weaknesses of our player modeling approach, particularly in replicating the driving style of the human player, and so we were surprised to find that its performances in these experiments were close to those of its human counterparts. In several instances in which one of the authors was testing a track tailored to the model of his own driving, he would obtain a final progress quite close to the target for which the track had been evolved. This typically happened only on the first trial on the new track; in subsequent attempts on the same track, the player would generally improve on his first trial by exploiting his accumulated knowledge of the track, an impossibility for a reactive controller.

In figure 2(a) we can see plotted the evolution of the three distinct fitnesses of the best individual throughout the evolutionary run (10 evolutionary runs are averaged) when only the final progress fitness is in the selection mechanism; figure 2(b) shows instead the evolution of the fitnesses when all of the fitnesses are employed in the selection. As can be seen, in figure 2(a) the progress fitness quickly converges to 1 in the first tens of generations. Top speed and final progress standard deviation instead reduce in favour of the final progress optimization. When optimization for top speed and progress variance is also enforced

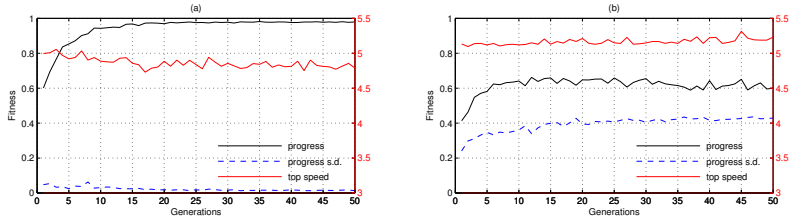

Fig. 2. Fitnesses of the best individual (average of 10 evolutionary runs): (a) selection based only on track progress finess, (b) selection based on all the three fitnesses.

(figure 2(b)), it seems clear that these fitness measures are in direct competition. The evolutionary algorithm drives the progress fitness toward the requirement, settling at fitness 0.6, a reasonable value since the final progress standard deviation is equal to 0.4. The initial population is constituted by straight tracks, the top speed is therefore close to the maximum since the early generations; the

evolutionary selection can be seen to guarantee the top speed not to be dropped in favour of the other fitnesses.

In figure 3 are displayed three tracks that evolution tailored on the player model of two of the authors; track ((*a*)) is evolved for a final progress of 1.1 (since the respective human player was not very skilled), track (*b*) and (*c*) are instead evolved on a model of a much skilled player for final progress [1] 1.5. For track (*a*) and (*b*) all the three fitness measure were used, while for track (*c*) only progress fitness was used.

The main difference between tracks (*a*) and (*b*) is that track (*a*) is broader and has fewer tricky passages, which makes sense as the player model used to evolve (*a*) drives slower. Both contain straight paths that allow the controller to achieve high speeds. In track (*b*) we can definitely notice the presence of narrow passages and sharp turns, elements that force the controller to reduce speed but only sometimes causes the car to collide. Those elements are believed to be the main source of final progress variability. These features are also notably absent from track *c*, on which the good player model has very low variability. The progress of the controller is instead limited by many broad curves.

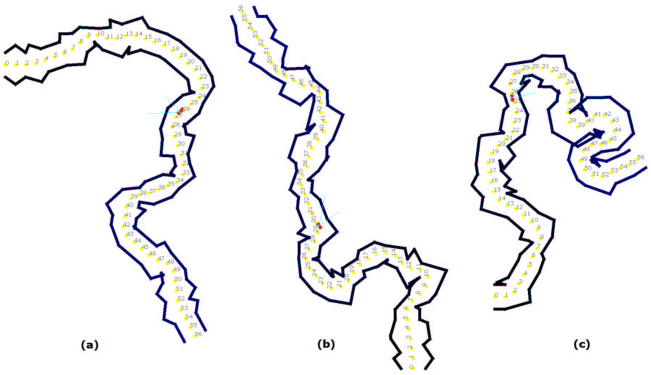

Fig. 3. Three evolved tracks: ((*a*)) evolved for a bad player with target progress 1.1, (*b*) evolved for a good player with target fitness 1.5, (*c*) evolved for a good player with target progress 1.5 using only progress fitness.

7 Conclusions

We have shown that we can evolve tracks that, for a given controller, will yield a predefined progress for the car in a given time, while maximizing the maximum

[1] The target progress is set between 50 and 75 percent of the progress achievable by the specific controller in a straight path. As a comparison, in Formula 1 races this ratio (calculated as ratio between average speed and top speed) is about 70 percent, and for the latest *Need for Speed* game it is between 50 and 60 percent.

speed and the standard deviation of the progress. The tracks produced are all drivable for a human player, and appear quite well designed, to our eyes at least. We do not yet know whether we have succeeded in automatically creating entertaining tracks for particular players, for the simple reason that we have not done any empirical studies on actual human players (apart from ourselves) to find out what they like. This is something that definitely needs to be done.

Our approach using player modeling works less well, but we can at least reliably produce controllers that make the same progress as the corresponding human player on a number of tracks, even though the driving styles differ from human driving. We found this level of modelling to be sufficient for track evolution, though of course we would like to be able to model human driving more closely. The failure of our supervised learning approaches to player modelling is probably due to the simplicity of the rangefinder sensors compared to human vision, and of the feedforward neural networks compared to human cognitive capacities, but additional experiments need to be done using recurrent neural networks and detailed visual inputs for the controllers to establish this.

Some interesting and relatively simple extensions of this research would be to evolve appropriate opponents in a multi-car race, and also to evolve tracks for learnability, following Koster's theory of what makes a game fun.

Acknowledgement Thanks to Owen Holland for stimulating discussions.

References

1. Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing. In: Proceedings of the Congress on Evolutionary Computation. (2005) 1906–1913
2. Togelius, J., Lucas, S.M.: Evolving robust and specialized car racing skills. In: Proceedings of the IEEE Congress on Evolutionary Computation. (2006)
3. Togelius, J., Lucas, S.M.: Arms races and car races. In: Submitted. (2006)
4. Kendall, G., Lucas, S.M., eds.: Proceedings of the First IEEE Symposium on Computational Intelligence and Games. IEEE Press (2005)
5. Yannakakis, G.N.: AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation. PhD thesis, School of Informatics, University of Edinburgh (2005)
6. Koster, R.: A Theory of Fun for Game Design. Paraglyph Press (2004)
7. Jirenhed, D.A., Hesslow, G., Ziemke, T.: Exploring internal simulation of perception in mobile robots. In: Proceedings of the Fourth European Workshop on Advanced Mobile Robots. (2001) 107–113
8. Charles, D., Black, M.: Dynamic player modelling: A framework for player-centred digital games. In: Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education. (2004) 29–35
9. Yannakakis, G., Maragoudakis, M.: Player modeling impact on players entertainment in computer games. In: Proceedings of the 10th International Conference on User Modeling. (2005) 74–78
10. Herbrich, R.: personal communication (2006)
11. Pomerleau, D.A.: Neural network vision for robot driving. In: The Handbook of Brain Theory and Neural Networks. (1995)