Towards automatic personalised content creation for racing games

Julian Togelius, Renzo De Nardi and Simon M. Lucas
Department of Computer Science
University of Essex, UK
{jtogel, rdenar, sml}@essex.ac.uk

Abstract— **Evolutionary algorithms are commonly used to create high-performing strategies or agents for computer games. In this paper, we instead choose to evolve the racing tracks in a car racing game. An evolvable track representation is devised, and a multiobjective evolutionary algorithm maximises the entertainment value of the track relative to a particular human player. This requires a way to create accurate models of players' driving styles, as well as a tentative definition of when a racing track is fun, both of which are provided. We believe this approach opens up interesting new research questions and is potentially applicable to commercial racing games.**

Keywords: Car racing, player modelling, entertainment metrics, content creation, evolution.

I. Three approaches to computational intelligence in games

Much of the research done under the heading "computational intelligence and games" aims to optimise game playing strategies or game agent controllers. While these endeavours are certainly worthwhile, there are several other quite different approaches that could be at least as interesting, from both an academic and a commercial point of view.

In this paper we discuss three approaches to computational intelligence in games: optimisation, imitation and innovation. We describe these approaches as they apply to games in general and exemplify them as they apply to racing games in particular. We then describe an experiment where these approaches are used in a racing game to augment player satisfaction. The taxonomy given below is of course neither final nor exhaustive, but it is a start.

A. The optimisation approach

Most research into computational intelligence and games takes the optimisation approach, which means that an optimisation algorithm is used to tune values of some aspect of the game. Examples abound of using evolutionary computation to develop good game-playing strategies, in all sorts of games from chess to poker to warcraft [1][2].

Several groups of researchers have taken this approach towards racing games. Tanev [3] developed an anticipatory control algorithm for an R/C racing simulator, and used evolutionary computation to tune the parameters of this algorithm for optimal lap time. Chaperot and Fyfe [4] evolved neural network controllers for minimal lap time in a 3D motocross game, and we previously ourselves investigated which controller architectures are best suited for such optimisation in a simple racing game [5]. Sometimes optimisation is multiobjective, as in our previous work on optimising controllers for performance on particular racing tracks versus robustness in driving on new tracks [6]. And there are other things than controllers that can be optimised in car racing, as is demonstrated by the work of Wloch and Bentley, who optimised the parameters for simulated Formula 1 cars in a physically sophisticated racing game [7].

While games can be excellent test-beds for evolutionary and other optimisation algorithms, it can be argued that improving game-playing agents is in itself of little practical value. From the point of view of commercial game developers, most game genres are not in a great need of more effective computer-controlled agents or strategies, as it is already easy to create competitors that beat all human players (though there are exceptions to this, such as real-time strategy games, where more effective AI is a hot research topic [2]). The problem is rather that game agents don't behave *interestingly* enough.

From the points of view of evolutionary roboticists, neuroscientists and other cognitive scientists, optimal behaviour is often uninteresting. Games can definitely be interesting as environments in which to study the emergence of intelligence or certain neural mechanisms, but this requires that both fitness function and environment allows for behaviours of the right complexity, and that the particular phenomena to be studied are not "abstracted away" by the non-computational intelligence parts of the game.

B. The innovation approach

The boundary between the optimisation approach and the innovation approach is not clear-cut, but the innovation approach is more focused on generating interesting, or complex, as opposed to optimal behaviour. The innovation approach sees games as environments for the development of complex intelligence, rather than computational intelligence techniques as means of achieving particular results as games. (Though the two perspectives are of course not exclusive, and many projects take both.) Typically this entails not knowing exactly what one is looking for.

In this approach, it is desirable not to constrain the creativity of the evolutionary algorithm, and, if evolving controllers, that the controller is situated within a closed sensorimotor loop [8]. Therefore, the agents are usually fed relatively unprocessed data (such as first-person visual or other sensor data) instead of abstract and pre-categorized representations of the environment (such as types and numbers of enemies around, or parameters describing the racing track), and the outputs of the controller are treated as primitive movement commands rather than e.g. which plan to select.

In car racing we can see examples of the innovation approach to computational intelligence in work done by Floreano et al. [9] on evolving active vision, work which was undertaken not to produce a controller which would follow optimal race-lines but to see what sort of vision system would emerge from the evolutionary process. We have previously studied the effect of different fitness measures in competitive co-evolution of two cars on the same tracks, finding that qualitatively different behaviour can emerge depending on whether controllers are rewarded for relative or absolute progress [10].

C. *The imitation approach*

While evolutionary computation is predominant in the two previous approaches, the imitation approach relies on various forms of supervised learning. Typically, what is imitated is a human player, but a game agent can of course plausibly try to imitate another agent.

A major example of the imitation approach to computational intelligence in racing games is the XBox game Forza Motorsport from Microsoft Game Studios. In this game, the player can train a "drivatar" to play just like himself, and then use this virtual copy of himself to get ranked on tracks he doesn't want to drive himself, or test his skill against other players' drivatars. Moving from racing games to real car driving, Pomerleau's work on teaching a real car to drive on highways through supervised learning based on human driving data is worth mentioning as an example of the imitation approach [11]. The reason for using imitation rather than optimisation in this case was probably not that interesting driving was preferred to optimal driving, but rather that evolution using real cars on real roads would be costly.

Our own work on imitating the driving styles of real human players

D. *Combining imitation and innovation for content creation*

All the above examples deal with designing or tuning behaviours and other aspects of agents, i.e. vehicles. But there are no obvious reasons why this should not be done with other aspects of racing games. Indeed, very large parts of the budget of a commercial game go into creating game content, such as levels, tracks, and artwork, and there is no reason why computational intelligence should not be brought to bear on this domain.

In this paper, we propose a method for on-line personalised automatic content creation, combining the imitation and innovation approaches. The first step of this method is to acquire a model of the human driver, which is accurate in relevant respects. The controller representation and racing game used for the modelling is the same as in our earlier experiments using the optimisation approach. Next, we evolve new racing tracks specifically tailored to the modelled human, using the controller generated through modelling to test the tracks. The tracks are "optimised" for entertainment value.

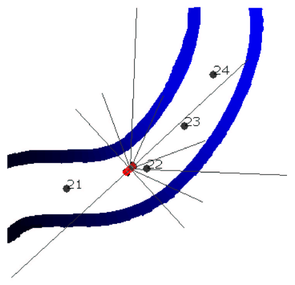

Fig. 1. Close-up of the car, and a section of the test track. Lines protruding from the car represent the positions and ranges of the wall sensors.

This paper is based on ideas and preliminary experiments reported in our earlier workshop paper [12]. In this paper, we extend the discussion, and report new and significantly different methods for both player modelling and track evolution.

II. Racing game, sensors, controllers

Our experiments make use of a simple racing game, which was developed in order to qualitatively reproduce the experience of driving a toy radio-controlled car on a tabletop track. The car actuators accept a discrete set of nine movement commands, corresponding to the keyboard arrow keys and combinations of them. A Newtonian physics model is implemented, allowing for momentum, skidding, and complex rebounds from collisions with walls or other vehicles. Apart from walls, tracks also consist of a number of waypoints, which the car must pass in order. In our previous experiments the fitness of the controller was computed as the number of waypoints passed in a certain period of time; below we use the waypoints in a slightly more sophisticated way.

The controllers are based on neural networks, and take sensor information as inputs and produces movement commands as outputs. As for the sensors, these consist of the speed of the car, a waypoint sensor giving the angle between the car's heading and the direction to the current waypoint, and a number of wall sensors. The wall sensors return the approximate distance to the nearest wall in a certain direction, or zero if no wall is within range. For the current experiments we use ten wall sensors on the car, ranges between 100 and 200 pixels and more sensors in the front of the car than in the back. All sensors are normalised to returning values between 0 and 1, and have a small amount of noise added to them.

III. The Cascading Elitism algorithm

We use artificial evolution both for modelling players and constructing new tracks, and in both cases we have to deal with multiobjective fitness functions. While evolutionary multiobjective optimisation is a rich and active research field, what we need here is just a simple way of handling more than one fitness function. We are not interested in pareto fronts; what we are interested in is specifying which fitness measures have higher priorities than others. A simple solution to this is using an evolution strategy with multiple elites. In the case of three fitness measures, it works as follows: out of a population of 100, the best 50 genomes are selected according to fitness measure f_1. From these 50, the 30 best according to fitness measure f_2 are selected, and finally the best 20 according to fitness measure f_3 are selected. Then these 20 individuals are copied four times each to replenish the 80 genomes that were selected against, and finally the newly copied genomes are mutated.

This algorithm, which we call Cascading Elitism, is inspired by an experiment by Jirenhed et al. [13].

1) On the effects of Cascading Elitism: At each generation this algorithm selects the elite on the basis of what is a non-linear combination of the fitness functions. If we consider the extreme case of two independent fitnesses, each one of the selection steps behaves like an independent elitist algorithm in which part of the elite is randomly removed (by the other selections). In another extreme case, that of inversely dependent fitnesses, the second selection step would always pick the worst part (the worst according to the first fitness but the best according to the second) of the first elite. The size of the first elite and the ratio between the two elites therefore starts to be important.

In our situation, like in most interesting problems, the fitnesses are neither independent nor fully inversely dependent, and a more in depth and systematic analysis is needed to go beyond mere speculations. While reserving this to future research, in our experiments the ratios $3/5$ and $2/3$ were arrived at through manual tuning.

IV. Player modelling

The first step in our method is to acquire a good model of the human driver, that can then be used to test tracks during evolution. Here, we first need to define what it means for a player model to be good, and then decide what learning algorithm and representation to use.

A. When is a player model adequate?

The only complete model of a human player is the human player himself. This is both because human brains and sensory systems are rather more complex than anything machine learning can learn, and because of the limited amount of training data available from the few laps around a test track which is the most we can realistically expect a player to put up with. Further, it is likely that a controller that accurately reproduces the player's behaviour in some respects and circumstances work less well in others. Therefore we need to decide what features we want from the player model, and which features have higher priority than others.

As we want to use our model for evaluating fitness of tracks in an evolutionary algorithm, and evolutionary algorithms are known to exploit weaknesses in fitness function design, the highest priority for our model is robustness. This means that the controller does not act in ways that are grossly inconsistent with the modelled human, especially that it does not crash into walls when faced with a novel situation. The second criterion is that the model has the same average speed as the human on similar stretches of track, e.g. if the human drives fast on straight segments but slows down well before sharp turns, the controller should do the same. That the controller has a similar driving style to the human, e.g. drives in the middle of straight segments but close to the inner wall in smooth curves (if the human does so), is also important but has a lower priority.

B. Direct modelling

What we call direct modelling is what is arguably the most straightforward way of acquiring a player model: use supervised learning to associate the state of the car with the actions the human take given that car state. We let several human players drive test tracks, and logged the speed and the outputs of waypoint sensor and the wall sensors (as defined above) together with the action taken by the human at each timestep. Two methods of supervised learning were tried on this data set: training a multilayer perceptron for use in the controller with backpropagation, and using the unprocessed data for controlling the car with nearest neighbour classification of input data. Both methods resulted in worthless controllers that rarely completed a whole lap. While the trained neural networks were worthless in an uninteresting way, the nearest neighbour-based controllers reproduced the modelled players' driving style almost perfectly, until the slight random perturbations in the game presented the controller with a situation that differed enough from anything present in the training data, and the car crashed. None of the controllers were able to recover from crashes, as the human players had not crashed during the data collection, and thus the situation was not in the data set.

We believe this not to be a problem with the particular supervised learning algorithms we used but rather an unavoidable problem with the direct modelling approach. As no model is perfect, controllers developed with direct modelling will tend to err, which diminish their performance to lower than the modelled human. In general, it is very unlikely that they will perform better than or as good as the modelled human (though it is theoretically possible that individual controllers could perform well), as any deviance from correct modelling will tend toward random behaviour. Such imperfect controllers will likely crash into walls, and will not know how to recover, as *the controllers can't learn from their mistakes*.

This problem was recognized by the developers of Forza Motorsport, who solved it by placing certain constraints on the types of tracks that were allowed in the game, and then recording the player's racing line over each possible track segment. Still, collisions with walls could not be entirely avoided, so a hard-coded crash-recovery behaviour was needed [14]. While this modelling method ostensibly works, it places far too many constraints on the tracks to be useful for our purposes.

C. Indirect modelling

Indirect modelling means measuring certain properties of the player's behaviour and somehow inferring a controller that displays the same properties. This approach has been taken by e.g. Yannakakis in a simplified version of the Pacman game [15]. In our case, we start from a neural network-based controller that has previously been evolved for robust but not optimal performance over a wide variety of tracks, as described in [6]. We then continue evolving this controller with the fitness function being how well its behaviour agrees with certain aspects of the human player's behaviour. This way we satisfy the top-priority robustness criterion, but we still need to decide on what fitness function to employ in order for the controller to satisfy the two other criteria described above, situational performance and driving style.

In our earlier paper [12], we measured the average driving speed of the human player on three tracks designed to represent different types of driving challenges, and then evolved controllers to match that performance as closely as possible on each of the three tracks. That method was successful, but could be argued to fail to capture much of the driving style of the player. Here we make an attempt to model the driving in more detail while still using an indirect approach.

First of all, we design a test track, featuring a number of different types of racing challenges. The track, as pictured in (fig 2), has two long straight sections where the player can drive really fast (or choose not to), a long smooth curve, and a sequence of nasty sharp turns. Along the track are 30 waypoints, and when a human player drives the track, the way he passes each waypoint is recorded. What is recorded is the speed of the car when the waypoint is passed, and the orthogonal deviation from the straight path between the waypoints, i.e. how far to the left or right of the waypoint the car passed. This matrix of two times 30 values constitutes the raw data for the player model.

The actual player model is constructed using the Cascading Elitism algorithm, starting from a general controller and evolving it on the test track. Three fitness functions are used, based on minimising the following differences between the real player and the controller:

- f_1: total progress (number of waypoints passed within 1500 timesteps),
- f_2: speed at which each waypoint was passed,
- f_3: orthogonal deviation was passed.

The first and most important fitness measure is thus total progress difference, followed by speed and deviation difference respectively.

D. Results

In our experiments, five different players' driving was sampled on the test track, and after 50 generations of the Cascading Elitism algorithm with a population of 100, controllers whose driving bore an acceptable degree of resemblance to the modelled humans had emerged. The total progress varied considerably between the five players - between 1.31 and 2.59 laps in 1500 timesteps - and this difference was faithfully replicated in the evolved controllers, which is to say

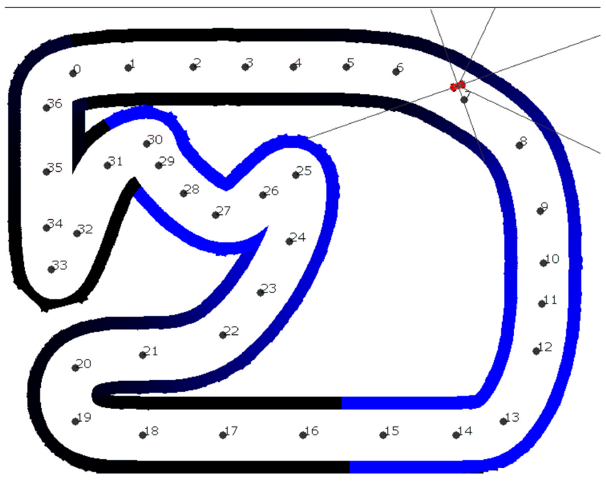

Fig. 2. The test track and the car.

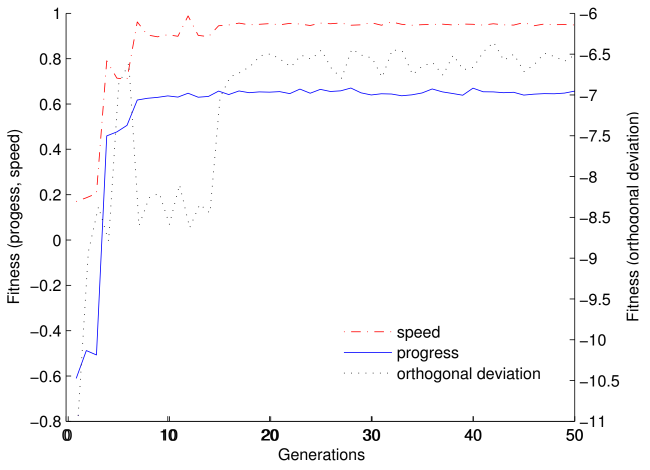

Fig. 3. Evolving a controller to model a slow, careful driver. Since the initial general controller is quite performing, the evolutionary algorithm quickly adapts the driving style to obtain the required progress and speeds. At last also the ortogonal deviation fitness improves. See IV-C for the description of the fitnesses.

that some controllers drove much faster than others (see the speed fitness in fig.3 and fig.4) . Progress was made on the two other fitness measures as well, and though there was still some numerical difference between the real and modelled speed and orthogonal deviation at most waypoint passings, the evolved controllers do reproduce qualitative aspects of the modelled players' driving. For example, the controller modelled on the first author drives very close to the wall in the long smooth curve, very fast on the straight paths, and smashes into the wall at the beginning of the first sharp turn. Conversely, the controller modelled on the anonymous and very careful driver who scored the lowest total progress crept along at a steady speed, always keeping to the center of the track.

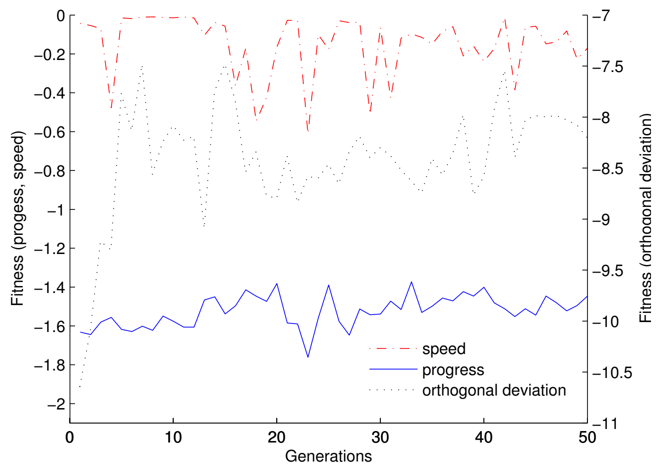

Fig. 4. Evolving a controller to model a good driver. The lack of progress on minimising the progress difference is the result of the fact that the progress of the modelled driver is very close to that of the generic controller used to initialise the evolution. See IV-C for the description of the fitnesses.

V. Track evolution

Once a good model of the human player has been acquired, we will use this model to evolve new, fun racing tracks for the human player. In order to do this, we must know what it is for a racing track to be fun, how we can measure this property, and how the racing track should be represented in order for good track designs to be in easy reach of the evolutionary algorithm. We have not been able to find any previous research on evolving tracks, or for that sake any sort of computer game levels or environments. However, Ashlock et al.'s paper on evolving path-finding problems is worthy to mention as a an example of an approach that could possibly be extended to certain types of computer games [16].

A. *What makes racing fun?*

It is not obvious what factors make a particular racing track fun to drive, or how to measure any such factors. While several researchers, notably Malone and Koster, have tried to explain why some games are more fun than others in the context of computer games in general, we are aware of no research on the particular genre of racing games. The following discussion is based on Malone, Koster and our own observations.

Thomas Malone claims that the factors that make games fun can be organized into three categories: challenge, fantasy, and curiosity [17]. The first thing to point out about challenge is that the existence of some sort of goal adds to the entertainment value. Further, this goal should not be too hard or too easy to attain, and the player should not be too certain about what level of success he will achieve. Translated to the context of racing game tracks, this ought to mean that the track should not be too easy or too difficult to drive, and that the track should encourage the player to try driving strategies that might work, and might not. These factors can be estimated by how close the mean speed of the player on the track is to a pre-set target speed, and as how variable this mean speed is between attempts or laps, respectively.

Games that include fantasy, according to Malone, "show or evoke images of physical objects or social situations not actually present". The sensation of being somewhere else, being someone else, doing something else. This is an important aspect of many racing games, but probably not one we can investigate in the graphically limited simulation we are currently using.

Malone's third factor is curiosity. He claims that fun games have an "optimal level of informational complexity" in that their environments are novel and surprising but not completely incomprehensible. These are games that invite exploration, and keeps the user playing just to see what happens next. It is not entirely clear how this insight can be transferred to the domain of racing games. It could be argued that tracks should be varied, combining several different types of challenges in the same track. It could also be argued that getting to drive a new track drawn from a limitless supply whenever you want, provokes enough curiosity, in which case the very method we are proposing in this paper is the answer to the curiosity challenge, as evolutionary algorithms are very good at coming up with unexpected solutions.

Raph Koster has a different take on fun, when he claims that fun is learning, and games are more or less fun depending on how good or bad teachers they are [18]. He concurs with Malone that the level of challenge in a game should be appropriate, but further claims that the game should display a good learning curve: new, more complex and rewarding challenges should be introduced at the rate old challenges are mastered. In the car racing domain this could mean that a good track design is one which is initially hard to drive, but which the player quickly learns to master.

An observation of our own, confirmed by the opinions of an unstructured selection of non-experts, is that tracks are fun where it is possible to drive very fast on straight sections, but it is necessary to brake hard in preparation for sharp turns, turns which preferably can be taken by skidding. In other words, it's fun to almost lose control. However, it is possible that this is a matter of personality, and that different players attach very different values to different fun factors. Some people seem to like to be in control of things, and people have very different attention spans, which should mean that some people would want tracks that are easier to learn than others. Identifying different player types and being able to select a mix of fun factors optimal to these players would be an interesting project, but we are not aware of any empirical studies on that subject.

B. *Fitness functions*

Developing reliable quantitative measures of, and ways of maximising, all the above properties would probably require significant effort. For this paper we chose a set of features which would be believed not to be too hard to measure, and designed a fitness function based on these. The features we want our track to have for the modelled player, in order of decreasing priority, is the right amount of challenge, varying amount of challenge, and the presence of sections of the track in which it is possible to drive really fast. The corresponding fitness functions are:

- f_1: the negative difference between actual progress and target progress (in this case defined as 30 waypoints in 700 timesteps),
- f_2: variance in total progress over five trials of the same controller on the same track,
- f_3: maximum speed.

C. Track representation

In our earlier paper we evolved fixed-length sequences of track segments. These segments could have various curvatures and decrease or increase the breadth of the track. While this representation had the advantage of very good evolvability in that we could maximise both progress and progress variance simultaneously, the evolved tracks did look quite jagged, and were not closed; they ended in a different point than they started, so the car had to be "teleported" back to the beginning of the track. We therefore set out to create a representation that, while retaining evolvability, allowed for smoother, better-looking tracks where the start and end of the track connect.

The representation we present here is based on b-splines, or sequences of Bezier curves joined together. Each segment is defined by two control points, and two adjacent segments always share one control point. The remaining two control points necessary to define a Bezier curve are computed in order to ensure that the curves have the same first and second derivatives at the point they join, thereby ensuring smoothness. A track is defined by a b-spline containing 30 segments, and mutation is done by perturbing the positions of their control points.

The collision detection in the car game works by sampling pixels on a canvas, and this mechanism is taken advantage of when the b-spline is transformed into a track. First thick walls are drawn at some distance on each side of the b-spline, this distance being either set to 30 pixels or subject to evolution depending on how the experiment is set up. But when a turn is too sharp for the current width of the track, this will result in walls intruding on the track and sometimes blocking the way. The next step in the construction of the track is therefore "steamrolling" it, or traversing the b-spline and painting a thick stroke of white in the middle of the track. Finally, waypoints are added at approximately regular distances along the length of the b-spline. The resulting track (see fig.2can look very smooth, as evidenced by the test track which was constructed simply by manually setting the control points of a spline.

D. Initialisation and mutation

In order to investigate how best to leverage the representational power of the b-splines, we experimented with several different ways of initialising the tracks at the beginning of the evolutionary runs, and different implementations of the mutation operator. Three of these configurations are described here.

1) Straightforward: The straightforward initial track shape forming a rectangle with rounded corners. Each mutation operation then perturbs one of the control points by adding numbers drawn from a gaussian distribution with standard deviation 20 pixels to both x and y axes.

2) Random walk: In the random walk experiments, mutation proceeds like in the straightforward configuration, but the initialisation is different. A rounded rectangle track is first subject to random walk, whereby hundreds of mutations are carried out on a single track, and only those mutations that result in a track on which a generic controller is not able to complete a full lap are retracted. The result of such a random walk is a severely deformed but still drivable track. A population is then initialised with this track and evolution proceeds as usual from there.

3) Radial: The radial method of mutation starts from an equally spaced radial disposition of the control points around the center of the image; the distance of each point from the center is generated randomly. Similarly at each mutation operation the position of the selected control point is simply changed randomly along the respective radial line from the center. Constraining the control points in a radial disposition is a simple method to exclude the possibility of producing a b-spline containing loops, therefore producing tracks that are always fully drivable.

E. Results

We evolved a number of tracks using the b-spline representation, different initialisation and mutation methods, and different controllers derived using the indirect player modelling approach.

1) Straightforward: Overall, the tracks evolved with the straightforward method looked smooth, and were just as easy or hard to drive as they should be: the controller for which the

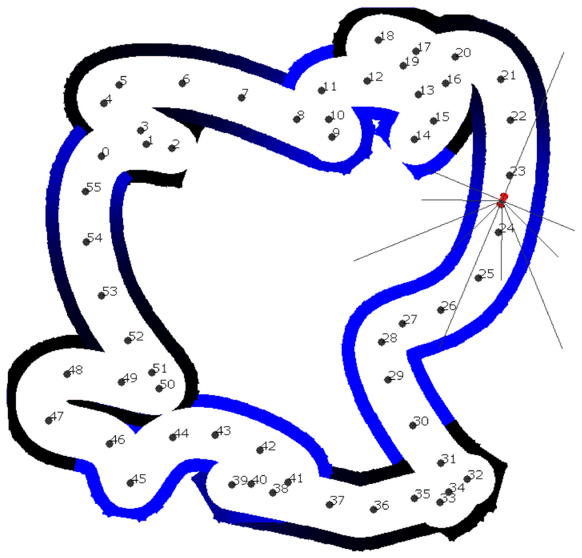
Fig. 5. Track evolved using the random walk initialisation and mutation.

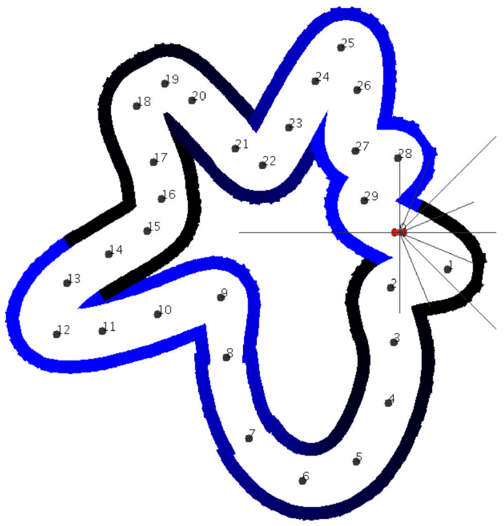

Fig. 6. A track evolved (using the radial method) to be fun for the first author, who plays too many racing games anyway. It is not easy to drive, which is just as it should be.

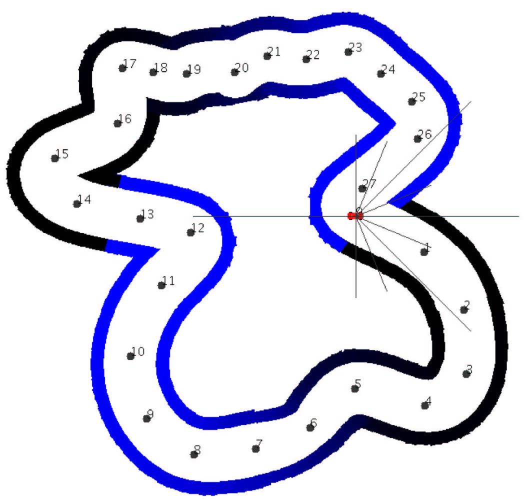

Fig. 7. A track evolved (using the radial method) to be fun for the second author, who is a bit more careful in his driving. Note the absence of sharp turns.

track was evolved typically made a total progress very close to the target progress. However, the evolved tracks didn't differ from each other as much as we would have wanted. The basic shape of a rounded rectangle shines through rather more than it should.

2) Random walk: Tracks evolved with random walk initialisation look weird (see 5) and differ from each other in an interesting way, and so fulfil at least one of our objectives. However, their evolvability is a bit lacking, with the actual progress of the controller often quite a bit different from the target progress and maximum speed low.

3) Radial: With the radial method, the tracks evolve rather quickly and look decidedly different (see fig.6 and 6 depending on what controller was used to evolve them, and can thus be said to be personalised. However, there is some lack of variety in the end results in that they all look slightly like flowers, clear bias of the type of mutation used.

4) Comparison with segment-based tracks: It is interesting to compare these tracks with some tracks evolved using the segment-based representation from our previous paper. Those tracks (see fig.8) do show both the creativity evolution is capable of and a good ability to optimise the fitness values we define. But they don't look like anything you would want to get out and drive on.

VI. Discussion

We believe the ideas described in this paper hold great promise, and that our player modelling method is good enough to be usable, but that there is much that needs to be done in order for track evolution to be incorporated in an actual game. To start with, the track representation and mutation methods need to be developed further, until we arrive at something which is as evolvable and variable as the segment-based representation but looks as good as (and is closed like) the b-spline-based representation. Features such as self-intersection also need to be allowed.

Further, the racing game we have used for this investigation is too simple in several ways, not least graphically but also in its physics model being two-dimensional. A natural next step would be to repeat the experiments performed here in a graphically advanced simulation based on an suitable physics engine, such as Ageia's PhysX technology [19]. In such a simulation, it would be possible to evolve not only the track in itself, but also other aspects of the environment, such as buildings in a city in which a race takes place. This could be done by combining the idea of procedural content creation [20][21] with evolutionary computation. Another exciting prospect is evolving personalised competitors, building on the results of our earlier investigations into co-evolution in car racing [10].

In the section above on what makes racing fun, we describe a number of potential measures of entertainment value, most of which are not implemented in the experiments described here. Defining quantitative versions of these measures would definitely be interesting, but we believe it is more urgent to study the matter empirically. Malone's and Koster's oft-cited hypotheses are just hypotheses, and as far as we know there are no psychological studies that tell us what entertainment metric would be most suitable for particular games and types of player. Real research on real players is needed. Such research could be in the vein of Yannakakis' and Hallam's studies on the Pac-Man game [22], where human players' reports on how much they enjoyed playing the game under various configurations were correlated with quantitative approximations of challenge and curiosity.

Finally we note that although we distinguished between different approaches to computational intelligence and games in the beginning to this paper, many experiments can be viewed from several perspectives. The focus in this paper on using evolutionary computation for practical purposes in games is not at all incompatible with using games for

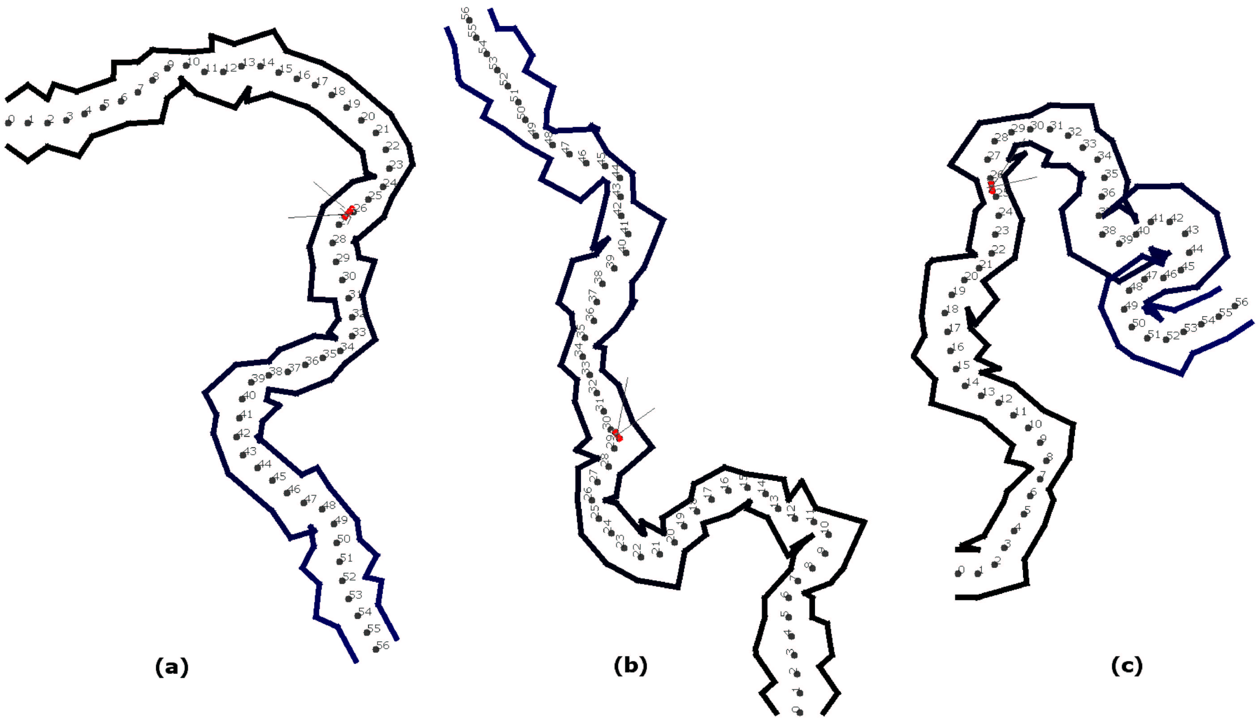

Fig. 8. Tracks evolved using the segment-based method. Track (a) is evolved for a weak player, and tracks (b) and (c) for a good player. Tracks (a) and (b) are evolved using all three fitness functions defined above, while track (c) is evolved using only progress fitness.

studying under what conditions intelligence can evolve, a perspective we have taken in some of our previous papers. On the contrary.

VII. Acknowledgements

We are thankful to Owen Holland, Georgios Yannakakis, Richard Newcombe and Hugo Marques for insightful discussions.

References

[1] G. Kendall and S. M. Lucas, *Proceedings of the IEEE Symposium on Computational Intelligence and Games*. IEEE Press, 2005.

[2] P. Spronck, "Adaptive game ai," Ph.D. dissertation, University of Maastricht, 2005.

[3] I. Tanev, M. Joachimczak, H. Hemmi, and K. Shimohara, "Evolution of the driving styles of anticipatory agent remotely operating a scaled model of racing car," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC-2005)*, 2005, pp. 1891–1898.

[4] B. Chaperot and C. Fyfe, "Improving artificial intelligence in a motocross game," in *IEEE Symposium on Computational Intelligence and Games*, 2006.

[5] J. Togelius and S. M. Lucas, "Evolving controllers for simulated car racing," in *Proceedings of the Congress on Evolutionary Computation*, 2005.

[6] ——, "Evolving robust and specialized car racing skills," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.

[7] K. Wloch and P. J. Bentley, "Optimising the performance of a formula one car using a genetic algorithm," in *Proceedings of Eighth International Conference on Parallel Problem Solving From Nature*, 2004, pp. 702–711.

[8] D. Cliff, "Computational neuroethology: a provisional manifesto," in *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, 1991, pp. 29–39.

[9] D. Floreano, T. Kato, D. Marocco, and E. Sauser, "Coevolution of active vision and feature selection," *Biological Cybernetics*, vol. 90, pp. 218–228, 2004.

[10] J. Togelius and S. M. Lucas, "Arms races and car races," in *Proceedings of Parallel Problem Solving from Nature*. Springer, 2006.

[11] D. A. Pomerleau, "Neural network vision for robot driving," in *The Handbook of Brain Theory and Neural Networks*, 1995.

[12] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.

[13] D.-A. Jirenhed, G. Hesslow, and T. Ziemke, "Exploring internal simulation of perception in mobile robots," in *Proceedings of the Fourth European Workshop on Advanced Mobile Robots*, 2001, pp. 107–113.

[14] R. Herbrich, "(personal communication)," 2006.

[15] G. N. Yannakakis and M. Maragoudakis, "Player modeling impact on players entertainment in computer games," in *User Modeling*, 2005, pp. 74–78.

[16] D. Ashlock, T. Manikas, and K. Ashenayi, "Evolving a diverse collection of robot path planning problems," in *Proceedings of the Congress On Evolutionary Computation*, 2006, pp. 6728–6735.

[17] T. W. Malone, "What makes things fun to learn? heuristics for designing instructional computer games," in *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, 1980, pp. 162–169.

[18] R. Koster, *A theory of fun for game design*. Paraglyph press, 2004.

[19] D. Gamez, R. Newcombe, O. Holland, and R. Knight, "Two simulation tools for biologically inspired virtual robotics," in *Proceedings of the IEEE 5th Chapter Conference on Advances in Cybernetic Systems*, 2006, pp. 85–90.

[20] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 2002.

[21] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation of 'pseudo infinite' cities," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 2003.

[22] G. N. Yannakakis and J. Hallam, "Towards capturing and enhancing entertainment in computer games," in *Proceedings of the Hellenic Conference on Artificial Intelligence*, 2006, pp. 432–442.