Generating Diverse Opponents with Multiobjective Evolution

Alexandros Agapitos, Julian Togelius, Simon M. Lucas, Jürgen Schmidhuber and Andreas Konstantinidis

Abstract— **For computational intelligence to be useful in creating game agent AI, we need to focus on creating interesting and believable agents rather than just learn to play the games well. To this end, we propose a way to use multiobjective evolutionary algorithms to automatically create populations of Non-Player Characters (NPCs), such as opponents and collaborators, that are interestingly diverse in behaviour space. Experiments are presented where a number of partially conflicting objectives are defined for racing game competitors, and multiobjective evolution of GP-based controllers yield pareto fronts of interesting controllers.**

Keywords: Genetic Programming, Reinforcement Learning, Multiobjective Evolution, AI in Computer Games, Car Racing

I. Introduction

When learning to play a game, the objective to maximize is usually taken to be a one-dimensional progress measure such as the score obtained by the agent, the skill of enemies that can be defeated, or the length of time the agent survives. Such measures come naturally to most computational intelligence researchers who are also gamers, as games usually judge the performance of a player based on one criteria only. What is a high-score list, if not a paradigmatic example of single-objective ranking?

This mode of thinking is appropriate when using games for testing computational intelligence (*CI*) algorithms, but not when developing CI techniques for use in games. This is because the proper role of CI (and other forms of AI) in a game is typically not to play the game well, but to provide interesting behaviour for *NPC*s (Non-Player Characters), such as opponents, competitors or sidekick.

An important explanation for the non-interest shown by the entertainment industry for most research in computational intelligence and games is that it is relatively easy for a game developer to construct an AI for a game that plays the game well. Or rather: even if it is not always straightforward to construct an AI that *actually* plays the game well, it is typically easy to cheat a little bit by providing the NPC with more information than the player, or better capabilities than the human-controlled agent, and thus achieve the desired level of competitiveness. For the most part, CI is just not needed here. (Though there are important exceptions, notably complex strategy games such as *Civilization*, where it is very hard to the design worthy opposition for a good human player without blatant cheating that threatens to dispel the player's suspension of disbelief.)

Alexandros Agapitos, Simon M. Lucas and Andreas Konstantinidis are with the Department of Computing and Electronic Systems, University of Essex, Colchester CO4 3SQ, United Kingdom. Julian Togelius and Jürgen Schmidhuber are with IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland. (emails: aagapi@essex.ac.uk, julian@idsia.ch, sml@essex.ac.uk, juergen@idsia.ch).

Much of the development effort when developing traditional game AI is instead focused on developing *interesting*, *diverse* and *believable* character AI. This is because even an opponent/collaborator that provides the adequate level of challenge/support for the human player may be very boring to play against/with, if it behaves in a very simple or repetitive and thus predictable way. In fact, predictable NPC behaviours is one of the most common complaints found under the "AI" heading in reviews of commercial games. Further, even if the individual NPCs have a reasonable range of behavioural responses, a game can quickly become boring if all the opponents or collaborators in a game behave in the same way. This is true for most genres of games, including real-time strategy (an army where all units respond the same way to new situations is not believable), first- and third-person shooters (it is no fun to be able to predict all your enemies' moves) and racing games (a starting field where all your competitors drive the same way doesn't require you to vary your driving style during the race).

Given the amount of work that goes into designing interesting, believable and especially diverse opponents, one of the principal ways in which computational intelligence could aid game designers is arguably through assisting in the design of such agents[1]. Some academic Computation Intelligence and Games researchers already focus on the problem of generating interesting NPC behaviours [2], [3]. This paper proposes a general approach to creating diverse and interesting NPC behaviours using multiobjective evolutionary algorithms (*MOEA*) in combination with a number of partly conflicting behavioural fitness measures. We do this by defining a number of such objectives for a car controller in a competitive racing game, and perform a number of experiments where we optimize for several of these objectives simultaneously using a standard MOEA. We then examine the pareto fronts resulting from the experiments, investigating to what extent we are able to automatically generate interestingly diverse controller populations. The vision is that a game designer, using this technique, should be able to automatically generate populations of opponents or collaborators spanning a range of interesting behaviours for any given game, environment, and possibly player.

A secondary motivation for these experiments is the observation that in some cases, even single-objective optimization might be aided by a multiobjective approach. This can be the case if the objective can somehow be decomposed into

[1]Another way is through the design of interesting game environments, which is the subject of [1], and yet another through the design of game rules, as discussed in another paper submitted to this conference.

several mutually reinforcing objectives. We have previously showed that for a version of the car racing task where the controller lacks a key input, the lack of which can be mitigated by good use of internal memory, an increase in attainability of the main objective (driving far on the track) can be achieved by adding a second "reinforcing" objective (use of internal memory) [4]. It is thus plausible that similar effects can be seen for some of the behavioural objectives used in this paper.

II. Methods

This section describes the car racing game used as a testbed in our experiments, the multiple fitness measures, and the multiobjective evolutionary algorithm used.

A. Car racing game

The experiments described below were performed in a 2-dimensional simulator intended to, qualitatively if not quantitatively model a standard radio-controlled (RC) toy car (approximately $17cm$ long) in an arena with dimensions approximately 3×2 meters, where the track is delimited by solid walls. The simulation has the dimensions $400 \times 300 pixels$, and the car measures $20 \times 10 pixels$.

In our model, a track consists of a set of walls, a chain of waypoints, and a set of starting positions and directions. When a car is added to a track in one of the starting positions, with corresponding starting direction, both the position and angle being subject to random alterations. The waypoints are used for fitness calculations.

The dynamics of the car are based on a reasonably detailed mechanical model, taking into account the small size of the car and bad grip on the surface, but is not based on any actual measurement [5][6]. The collision response system was implemented and tuned so as to make collisions reasonably realistic after observations of collisions with our physical RC cars. As an effect, collisions are generally undesirable, as they may cause the car to get stuck if the wall is struck at an unfortunate angle and speed. Just like most toy RC cars, the control of the car is bang-bang, with three possible drive modes (forward, backward, and neutral) and three possible steering modes (left, right and center).

Variations of racing games based on this simulator have been used as testbeds in a string of papers in recent years, as well as in two competitions associated with international conferences. For an overview, and a more detailed description of the racing simulator, see [7]. In particular, human-competitive neural network-based controllers for a single car on a single track were evolved in [8]; general controllers capable of driving on a wide range of tracks were evolved in [9]; co-evolution of two cars on a single track was explored in [10]; and controller based on genetic programming rather than neural networks were evolved in [11].

B. GP-based controllers, inputs and outputs

The controllers employ an expression-tree representation as practiced in standard functional Genetic Programming. Details on the GP system used can be found in [11].

For programming language standard arithmetic and trigonometric functions have been defined. Selected elements of the state are available to the controller via formal parameters to the program. The available information is all such that it could in principle have been gathered by sensors placed on the car ('first person'): speed of the car, angle and distance to the next way point and distance to a wall or an opponent car

in a given direction relative to the heading of the car. A small amount of noise is added to all sensor readings.

As for the outputs of the controller, these are two real numbers which are interpreted by the simulation as any of the nine possible commands. The first controller output is interpreted as the command for driving forward if its value is above 0.3, backward if below -0.3 and neutral otherwise. The second output is interpreted as steering left, right or centre in the same manner.

C. Behavioural fitness measures

The original car racing experiments used a single fitness measure, namely how many way points were passed in 700 time steps. We call this *absolute progress fitness*. In [10], where we introduced another car on the same track, we experimented with a second fitness measure, *relative* progress fitness: how far ahead of the other car the controlled car was after 700 time steps. We found that controllers evolved with relative progress as the only fitness measure drove much more aggressively than those evolved for absolute progress, and often focused on pushing their opponents of the track more than on progressing along the course; they were also quite worthless in the absence of an opponent. Relative and absolute progress is thus an example of partly conflicting objectives. We also experimented with mixed fitness functions, with parts absolute fitness and part relative fitness, and found that we could modulate the aggressiveness of the resulti

In the experiments below, we use the following setup: the controlled car (the one whose controller is evaluated) is placed at random in one of two possible starting positions at the beginning of the race, subject to a small amount of noise in initial position and orientation. Another car (the competitor) is placed in the other starting position nearby and controlled during the race by an incrementally evolved general controller (see [9]) for details. The controller for the competitor car does not change during either evolutionary time or the course of a single trial.

Each fitness evaluation consisted of five independent races. Each race went on for 700 time steps, and during this time a number of key statistics on the behaviour of the controlled car were gathered. These statistics made it possible to calculate the ten fitness measures described below. For some of the objectives, we include our hypothesis about with which other objective(s) it conflicts.

1) **Absolute progress** is the "orthodox" and most straightforward fitness measure used in most of the experiments in evolutionary car racing: a continuous approx-

imation of the number of way points passed in 700 time steps.

2) **Relative progress** is the absolute progress of the controlled car, minus the absolute progress of the competitor. According to previous results in [10] there is a conflict between absolute and relative progress fitness, as it often pays off better to stop the competitor by pushing it into a wall as soon as possible rather than just driving as fast as possible.
3) **Maximum speed** is simply the maximum speed of the controlled car at any point during the race. The relationship to absolute and relative progress is not obvious: on the one hand, a car that has absolute progress close to zero obviously also has low maximum speed. On the other hand, a controller that only outputs the "accelerate" command and thus drives as fast as possible along the first straight segment of the track and crashes into the wall at the end of it has a high maximum speed but low absolute and relative progress.
4) **Progress variance** is the standard deviation of absolute progress fitness between the five trials that constitute each fitness evaluation. This can be seen as a measure of the boldness of a driving style.
5) **Number of steering changes** is the number of time steps minus the number of times the steering command changes between left, right and centre divided by the number of time steps (the number of steering changes is maximised by minimising this fraction and minimised by maximising it). Previous experience has shown that evolved neural network and GP drivers tend to oscillate quickly between different driving commands (thus having a low steering changes fitness) whereas humans change steering direction much less often.
6) **Number of driving changes** is (the number of time steps minus the number of times the acceleration command changes between accelerate, brake/backward and neutral) divided by the number of time steps.
7) **Wall collisions** is defined as (the number of time steps minus the number of times the controlled car collides with a wall) divided by the number of time steps (the number of wall collisions is maximised by minimising this fraction and minimised by maximising it). While it is expected that wall collision fitness will be in conflict with both progress variance and maximum speed, it is very unclear what relation it will have to absolute and relative progress fitness.
8) **Proximity to competitor** is calculated as 500 minus the number of pixels between the center of the controlled car and the competitor, averaged over all time steps. A high proximity to competitor means that the controlled car is either staying right behind, in front of or besides the competitor, or that both cars crashed next to each other.
9) **Car collisions (maximum)** is simply the number of car-to-car collisions that occured during the race. This should be positively correlated with proximity to competitor, but in conflict with absolute progress and number of steering and driving changes, as car-to-car collisions typically require corrective actions in order to avoid ensuing wall collisions.
10) **Car collisions (minimum)** is the number time steps minus the number of car-to-car collisions. This objective exists as it makes sense to both maximize and minimize the number of such collisions.

D. Multiobjective Evolutionary Algorithm, Variation Operators and Run Parameters

For multiobjective evolutionary algorithm we used the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [12]. The algorithm uses tournament selection with a tournament size of 7. In order to allow for more exploitation towards the end of each evolutionary run the tournament size has been made dynamic during the final 10 generations incremented by a percentage of 20% in each generation. The evolutionary run proceeds for 50 generations and the population size is set to 500 individuals. Evolution halts when all of 50 generations have elapsed. Ramped-half-and-half tree creation with a maximum depth of 8 is used to perform a random sampling of program space during run initialisation. During the run, expression trees are allowed to grow up to depth of 17. Heuristic search employs a mixture of mutation-based variation operators similarly to [13].

III. Experimental Methodology

For the purposes of car racing we define two different general kinds of behaviours :

(a) Aggressiveness, is an umbrella term that encompasses speed, wall and car collisions. While wall and car collisions have been already discussed in previous sections, speed levels can create a significant burden by requiring a car to flexibly avoid slowly moving opponents (especially when the cars are moving into narrow parts of the track).

(b) Opponent Weakness Exploitation, not a crisply defined term at this stage of our research but it generally concerns all those behaviours that can exploit mistakes made by opponent drivers. As an example consider a controller that learns how to take close turns often pushing away an opponent that takes wider turns.

The following section presents the results of our attempts to optimise genetically programmed controllers to exhibit the bahavioural characteristics described above. For purposes of clarity of presentation, pareto fronts that combine more than two objectives have been decomposed into pairs of objectives and these have been plotted in the cartesian space.

We have previously attempted [11] to understand and distil (by static inspection of expression trees) the inner workings of genetically programmed controllers, unfortunately, with little success. In this vein, we will attempt to shed more light into the way controllers operate by (a) performing a genotypic analysis and identifying the average use of parameterised sensor readings within the expression-trees; (b) calculating Pearson correlation coefficients between the values of driving/steering commands issued and the values of sensor readings in each time-step throughout a race (correlation coefficients are based on the average of 50 independent races of a controller) and (c) examining a series of scatter plots between driving/steering commands and selected sensor readings.

IV. Results

A. Optimisation for Aggressiveness

1) Optimising for wall collisions: The first step was to evolve controllers that learn how to keep a safe distance to the walls at all times, thus, simulating the behaviour of a conservative driver. An intuitive hypothesis suggests that this could be achieved by maximising *absolute progress* and minimising *wall collisions*. Much to our dismay, the desired objectives failed to be optimised and the evolved controllers exhibited a rather aggressive behaviour by developing high speeds and crashing into the walls while turning (see Figure 2(b)). The pareto front presented in Figure 1(a) clealy identifies this trend. Note the negative correlation in which a high absolute fitness is seen with low wall collisions fitness (high number of wall collisions).

The next step was to incorporate more objectives crafted to describe the frequency of changes in the steering and driving commands issued by the controller. We have previously observed that changes on these commands have been made quite frequently in evolved controllers whereas human drivers rely on a more steady style of driving, so, we decided to minimise the number *steering* and *driving changes* (the minimisation of steering and driving changes has been performed into different runs) combining them with the maximisation of *absolute progress* and minimisation of *wall collisions*. The pareto front resulting from the three objectives (steering changes, absolute fitness and wall collisions) has been decomposed into three pairs of objectives and is illustrated in Figures 1(b), 1(c),1(d). Surprisingly, we observed that high absolute fitness has been traded-off with low steering changes fitness (high number of steering changes) and low wall collision fitness (high number of wall collisions). Interestingly, in Figure 1(c) we note that that steering changes and wall collisions are non-conflicting objectives (low number of steering changes is seen with low number of wall collisions – although their relationship does not appear to be completely linear). At first sight it seems that a steady human driving style has not been adopted by the evolved controllers, resulting mainly in aggressive driving behaviours that exhibit numerous wall collisions in an effort to achieve high absolute fitness. On the other hand, a different trend has been observed into the majority of pareto fronts resulted from the optimisation of driving changes, absolute progress and wall collisions (an illustrative pareto front is presented in Figures 1(e), 1(f), 1(g)). In Figure 1(e) we note that high absolute progress fitness has been traded-off with a high driving changes fitness (low number of driving changes) and most importantly Figure 1(f) shows that in the majority of pareto front points, low wall collisions fitness (many wall collisions) correlates (with a non-linear relationship) with high driving changes fitness (small number of driving changes), an observation indicative of two conflicting objectives. Nevertheless, similarly to the previous case, there is a negative correlation between wall collision fitness and absolute progress fitness (Figure 1(g)) indicating that a controller that drives as far as possible will have to trade this off with a high number of wall collisions.

It has become apparent that in order to evolve controllers that drive conservatively without crushing into the walls we need to request the maximisation of driving changes (minimisation of driving changes fitness). Intuitively, a controller could avoid wall collisions by either constantly oscillating between forward and backward driving commands thus achieving a constant low speed or, at the best case, accelerate and brake only when appropriate. The next experiment has been setup in this way using three objectives: *absolute progress fitness*, *driving changes fitness*, *wall collision fitness* (progress and collision fitness to be maximised). A resulted pareto front is depicted in Figures 1(g), 1(h), 1(i). First thing to observe is a great diversity of points. Our hypothesis that a minimisation of driving changes will result in more diverse driving behaviours was justified when we tested the evolved controllers. In this case it is not very obvious that high absolute fitness is seen with a great number of driving changes (Figure 1(g)). In addition, Figure 1(h) shows that the relation between wall collision fitness and driving changes fitness is highly non-linear. The movement trace depicted in Figure 2(c) shows a smooth trajectory of the learner (red car) without any wall collisions, however, the car drives at a constant low speed and does not reach a high absolute progress fitness.

Figures 3(a) and 3(b) show the average use of formal parameters, representing sensor readings, in the controllers of pareto fronts generated by maximising and minimising the number of driving changes respectively. In Figure 3(a) we note the *angle to next way point* and *speed* sensor readings are the dominant parameters used by the program structures. Interestingly, the *angle to next way point* sensor reading is under dominant usage and significantly determines the steering direction (see a consistent negative correlation between steering commands and AWP in cases 1 and 2 of Table I). Car sensor readings for reasons that will be clear later on are not used at all. *Case 1* in Table I refers to the maximisation of driving changes and details a negative correlation between driving commands and speed sensor readings (i.e. low driving command when high speed is reached) indicative that the controller quickly oscillates between different driving commands in order to keep a steady speed and avoid wall collisions. On the other hand, we note that in *Case 2* of Table I the relation between speed sensor reading and driving command is positive explaining the fact that the controller can reach high speeds (aggressive behaviour, no avoidance of wall collisions). The examination of scatter

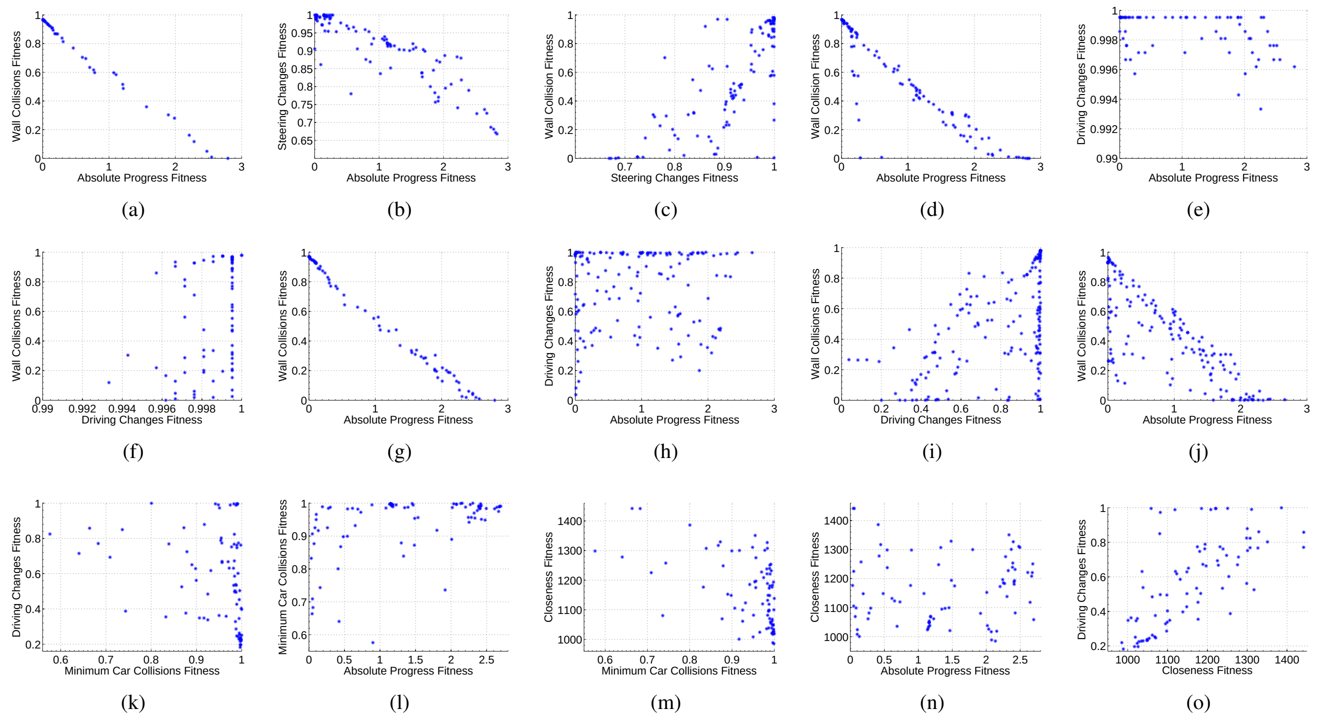

Fig. 1. Pareto Fronts: (a) optimising for wall collision avoidance; (b, c, d) optimising for aggression and max. speed; (e, f, g) optimising for aggression and max. speed; (h, i, j) optimising for smoothness, wall collision avoidance and low speed; (k, l, m, n, o) for maximum car collisions.

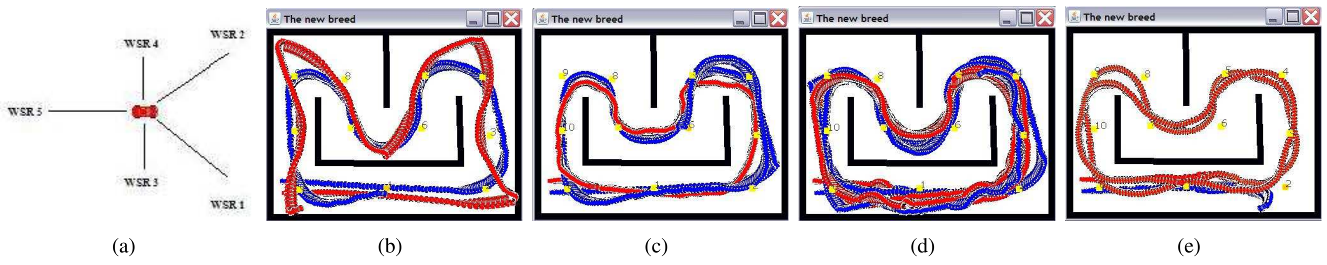

Fig. 2. (a) Wall-sensors setup (Car sensors have same orientation); Movement Traces (learner in red, opponent in blue): (b) maximum speed, wall-collisions; (c) low speed, no wall collisions; (d) car-collisions-provoking driver; (e) opponent weakness exploitation.

plots in Figures 4(a), 4(b) (referring to maximisation of driving changes) show that the controller is issuing a wider range of driving commands (including neutral) allowing it to better regulate its speed whereas Figures 4(c), 4(d) (referring to minimisation of driving changes) detail that the controller is exclusively issuing either forward or backward commands. A similar trend is depicted in the scatter plots of steering commands in which the wall avoiding controller issues numerous neutrals (no steering) allowing it to better regulate and smooths its orbit. The aggressive controller is mainly issuing left or right (Figure 4(d)) indicative of "desperate" efforts (mainly due to high speed) to get into course and orbit the next way point. However, this was a slow driver. Note how in the case of a quicker driver, in Figure 4(e), there are less neutral driving commands issued and these are issued in higher speeds. Also we observe a higher variance in angle-to-next-way-point values where the controller issues neutral steering commands.

2) Optimisation for car collisions: The next step was to evolve controllers that maximise the car collisions with the opponent driver. The combination of objectives that induced behaviourally interesting individulas included the maximisation of *absolute progress fitness*, *car collisions*, *car closeness*, and *driving changes*. Figure 3(d) shows the average use of formal parameters in the evolved expression trees of one of the most diverse pareto front resulted in an experimental run. We note that car-sensors are not being utilised by the evolved programs. In an attempt to understand why we recorded the car-sensor readings during 100 races of 10 different evolved controllers from different pareto fronts optimised using different objectives. Surprisingly, it turned out that opponent cars are most of the times invisible to the learner. Significant sensor values indicating the presence of a car appear only in the 100 time-steps where the competing cars are moving in the first straight track segment and are more likely to be moving next to each other. While the

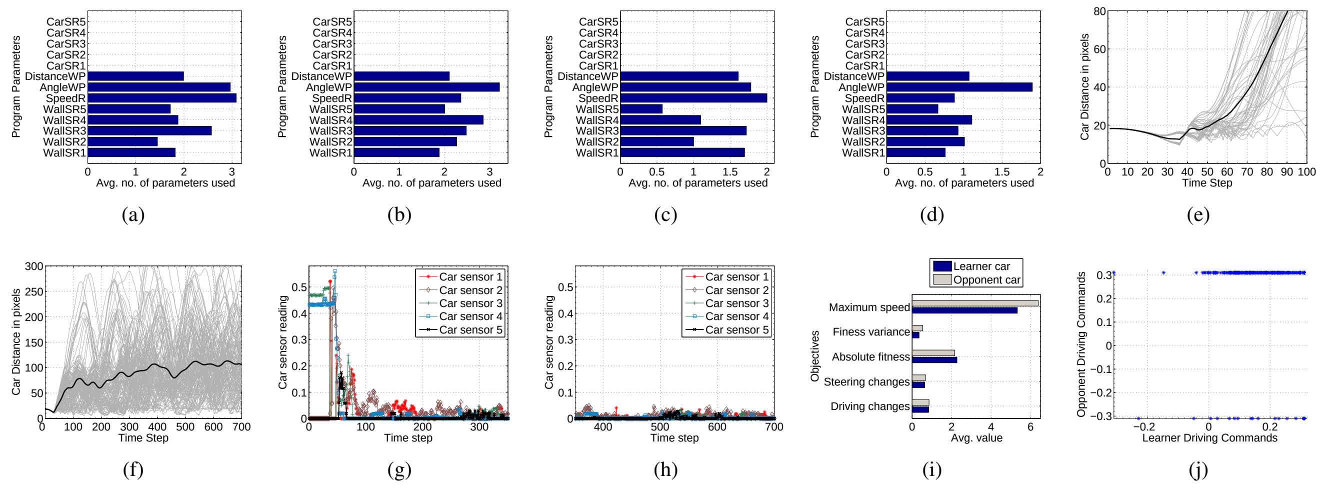

Fig. 3. (a, b, c, d) average usage of formal parameters in expression-trees; (e) distance between competing cars during first 100 time-steps in weakness exploitation behaviour (avg. of 50 races); (f) distance between competing cars during whole race (700 time-steps) in car-collisions-provoking behaviour (avg. of 50 races); (g, h) opponent car sensor readings during whole (700 time-steps) race averaged over 10 evolved controllers in 10 races with each one; (i) comparison of average values of behavioural objectives between opponent and a car-collisions-provoking driver; (j) scatter plot between driving commands issued in one sample race with a a car-collisions-provoking driver.

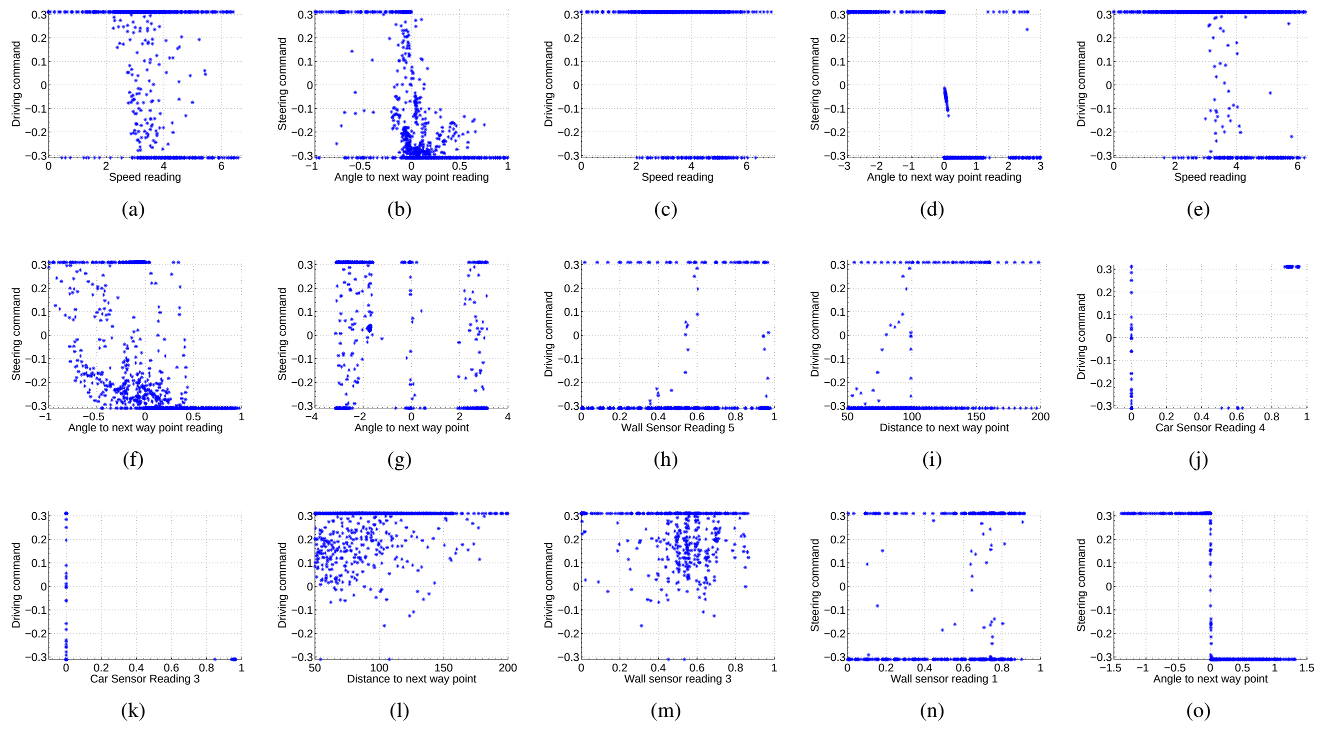

Fig. 4. Scatter plots of issued driving and steering commands against sensor readings throughout a sample race: (a, b) slow driver, no wall collisions; (c, d) wall colliding driver; (e, f) fast driver, no wall collisions; (g, h, i, j, k) opponent-weakness exploiter; (l, m, n, o) car-collisions-provoking driver.

graphs show an average of sensor values its variance is high (not shown for clarity) making these kind of sensor readings not a very reliable and consistent measurement for the learner. This is intuitive in a way if we consider that most of the times the distance between the cars oscillates making often making them invisible to each other. The testing of evolved controllers under this setup revealed that in order to maximise the car collisions, the learner needs to model the driving behaviour of the opponent and drive close to it at all times. The closer the driving distance the higher the likelihood of collision. Figure 3(f) show that for a car-collision-provoking controller the average distance between cars remains approximately constant throughout a race (avg. of 50 races). Also, for the same controller, Figure 3(i) shows its behavioural measurements matching those of the opponent car. Figure 3(j) illustrates an excellent relation between the driving commands issued by the opponents in the race depicted in Figure 2(e) (note the small course deviation that indicate car collisions). It is clear that the majority of times that the opponent issues a forward command the learner issues either forward or neutral in order to adjust its speed. The pareto front depicted in Figures 1(k), 1(l), 1(m), 1(n), 1(o) shows no correlation between closeness and minimum car collisions resulting in great diversity of values that does not necessarily reflect behavioural diversity. Interestingly, small number of driving changes is seen with high closeness between cars. On the other hand, driving changes did not seem to correlate with the number of car collisions and a widespread pareto front has been generated. Finally, high diversity is also noticed between absolute progress fitness and car-closeness, nevertheless, absolute progress is not a significant objective as the ultimate goal has been revealed to be the modeling of competitor's behaviour.

B. *Optimisation for Opponent Weakness Exploitation*

This is a rather obscure definition of behaviour, thus, the objectives that needed to be in place in order for the emergence of something interesting were not obvious prior to experimentation. Our intuition suggested that we could allow the evolutionary process to help us understand what could be possibly defined as *opponent weakness exploitation*. Indeed, a driving behaviour that was learned along this line was obtained while maximising for absolute progress, speed, closeness and steering changes. A movement trace of this bahaviour is illustrated in Figure 2(e). An opponent car (in blue) that ignores a approaching learner (red car gradually approach by the side) and keeps on moving in a straight line (without any attempt of avoidance) will be pushed away. Figure 3(e) plots the distance between the two cars for the first 100 time steps as recorded in 50 independent races (average shown with bold line). A close look reveals that the distance decreases after time-step 20 until it reaches a global minimum at around time-step 32 (when the impact takes place) and then gradually increases again leaving the opponent stuck against the wall in most of the times. The fact that the opponent car (in blue) constantly drives in a straight line is indicative that the learner is slowly approaches until it crushes onto the opponent and quite surprisingly proceeds counter-clockwise but facing the wrong way.

Figure 3(d) shows the average use of parameters within the expression-trees of the evolved pareto front. Surprisingly, the controllers, similarly to previous cases, make no use of car sensor readings. So, how can a learner know how to approach the opponent if that car is not visible? Looking for correlations in *Case 4* of Table I we noted a positive correlation between the driving command and car sensor readings 3 and 4 (these are sensors protruding vertically to the sides of the car – see Figure 2(a)). We then examined the series of scatter plots between the driving commands issued and the value of these two car sensors (Figures 4(j), 4(k)). The data for these scatter plots have been generated during a race where after the initial collision the learner process but faces the wrong way. Surprisingly, we found that for sensor 4, in Figure 4(k)), the controller issues a backward driving command while it senses the opponent. This happens towards the half of the first straight segment of the track. This is of course imaginary, the learner makes no use of car sensor readings, it is being discussed for the shake of demonstration of driving commands issued when the learner is in parallel to its opponent. The scatter plot (Figure 4(k)) indicates that as long as the learner is in parallel to the opponent it issues slowing commands to adjust its speed for collision. After the collision the opponent was crushed against the wall and the learner proceeded facing the wrong way, thus the positive driving command in Figure 4(j) (causing the car to slow down) each additional time the learner was passing by the stopped opponent. We argue that besides the fact that the learner was unable to see the opponent it was still possible to evolve a quite aggressive behaviour that was observed towards the middle of the first straight segment of the track and was only due to the commands emerging by the non-linear combination of way-point and wall sensor readings. The commands issued at that particular track segment often resulted in opponent knock out[2], thus, making the race easier and was often rewarded by selection pressure.

V. Discussion

Multiobjective evolutionary algorithms have not been applied widely to games so far; one very recent exception is due to Schrum and Miikkulainen [14]. In this paper, we have tried to demonstrate a way in which MOEAs lend themselves to improving the relevance of game learning research, by allowing us to create agents that not only play a game well, but in an interesting way.

A conceivable criticism of this idea is that it might not be very general: it works for car racing, but does it work for real-time strategy, first-person shooters and chess? We are still waiting for those experiments to be done, but there are reasons to believe it would work. Many of the objectives defined in this paper can be transformed more or less straightforwardly to other game genres. Absolute progress is simply the score of a game, or the number of captures or frags or some such measure; relative progress is simply absolute progress minus the absolute progress of your competitor or opponent (most games are not zero-sum games for most measurable quantities). Proximity to competitor is a measure that can be used in any game that takes place in physical space. The number of driving and steering changes objectives can be applied as they are to a game with discrete action space, or as an average control signal magnitude in games with continuous space.

Of course, there will always be a objectives that are unique for particular games or game genres. Examples of these (to be maximized or minimized) could be number of bullets fired or time spent hiding in a first-person shooter, number of resources secured or time from start of the game until the

[2]During that segment the opponent's trajectory is predictable and of course the two cars start in a high proximity, making it easier for the learner to approach and overtake.

TABLE I

PEARSON CORRELATION COEFFICIENTS BETWEEN DRIVING/STEERING COMMANDS AND SENSOR READINGS ISSUED EACH TIME-STEP AVERAGED OVER 50 INDEPENDENT RACES; *Case 1:* NON-WALL COLLISIONS; *Case 2:* WALL COLLISIONS, MAX. SPEED; *Case 3:* MAX. CAR COLLISIONS; *Case 4:* OPPONENT WEAKNESS EXPLOITATION

		WSR1	WSR2	WSR3	WSR4	WSR5	SR	AWP	DWP	CSR1	CSR2	CSR3	CSR4	CSR5
Case 1	Driving	.187	.016	−.173	−.057	.230	−.329	−.182	−.048	−.007	−.002	.061	.050	−.035
	Steering	.382	−.284	.197	−.218	.009	−.224	−.431	.096	−.006	−.011	−.008	.056	.024
Case 2	Driving	.043	.242	−.265	.079	.450	.568	.040	−.004	−.002	−.029	.098	.098	−.021
	Steering	.291	−.280	.124	−.127	−.036	−.122	−.628	.031	.013	.013	.010	.042	.006
Case 3	Driving	.039	.001	−.302	−.168	.347	−.439	.096	.223	−.045	−.016	−.003	.035	−.114
	Steering	.271	−.217	−.050	.022	.004	−.031	−.570	.004	.038	.006	.015	.046	.014
Case 4	Driving	.055	.270	−.234	.063	.256	.773	.106	.287	.057	.033	.533	.517	−.005
	Steering	.083	−.088	.012	−.044	.112	−.025	−.290	−.046	−.014	−.011	−.041	−.008	.020

first military encounter in a real-time strategy game, or the dispersion of units over the board or time until first capture in chess. The important thing is that the objectives should be at least partly conflicting for a pareto curve to be generated from which interesting strategies can be picked.

An interesting future research topic would be to automatically define new objectives. This could probably be done using statistical and clustering techniques, based on the behaviours of controllers of varying performance.

Multiobjective evolution with behavioural objectives could also be used to improve modelling of human playing styles. In [1] we argue (and exemplify) that direct modelling of human playing styles tends to result in player models that generalizes badly to new environments. It seems plausible that a combination of objectives based on consistent performance across multiple environments and objectives based on faithful replication of human playing styles could help in learning behaviour that was both robust and human-like. In the context of the current racing game, the fitness measures used in this paper could be complemented with e.g. average absolute progress on a number of tracks, variance in absolute progress on the same set of tracks, and similarity of driving to the recorded human player's driving on a test track (based on e.g. speed and lateral displacement at each way point).

In this paper, we have only investigated evolving against a single fixed opponent. It would also be interesting to evolve against a number of other cars and to evolve against cars controlled by models of human players, like in [1].

VI. CONCLUSIONS

We have argued that it is important for CIG research to focus on agents that not only play games well, but also that behave in interesting way. One way to automatically create such agents is to evolve populations that behave differently to each other along interesting dimensions, and then select various individuals from the population that are sufficiently dissimilar to each other. This can be done using multiobjective evolutionary algorithms and multiple partly conflicting behavioural fitness measures.

We have provided an example of this approach, through defining a number of suitable behavioural fitness measures for a car racing game and evolving neural network controllers for the game using two or three objectives at a time. Our experimental results show that a surprisingly rich repertoire of different strategies can be automatically generated using these simple means and an apparently simple game. They further showed that the interactions between the behavioural objectives produced unexpected effects which added to our understanding of the central mechanic of the game. We believe the technique presented in this paper to be useful for a large number of game genres, and even the specific behavioural fitness measures presented here to be transferrable to games of other genres.

REFERENCES

[1] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation in racing games," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[2] B. D. Bryant, "Evolving visibly intelligent behavior for embedded game agents," Ph.D. dissertation, Department of Computer Sciences, University of Texas, Austin, TX, 2006.

[3] G. N. Yannakakis, "Ai in computer games: Generating interesting interactive opponents by the use of evolutionary computation," Ph.D. dissertation, University of Edinburgh, 2005.

[4] A. Agapitos, J. Togelius, and S. M. Lucas, "Multiobjective techniques for the use of state in genetic programming applied to simulated car racing," in *Proc. of IEEE CEC*, 2007, pp. 1562–1569.

[5] D. M. Bourg, *Physics for Game Developers*. O'Reilly, 2002.

[6] M. Monster, "Car physics for games," http://home.planet.nl/ monstrous/tutcar.html, 2003.

[7] J. Togelius, "Optimization, imitation and innovation: Computational intelligence and games," Ph.D. dissertation, Department of Computing and Electronic Systems, University of Essex, Colchester, UK, 2007.

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

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

[10] ——, "Arms races and car races," in *Proceedings of Parallel Problem Solving from Nature*. Springer, 2006.

[11] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing using object oriented genetic programming," in *Proceedings of the Genetic and Evolutionay Computation Conference*, 2007.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[13] K. Chellapilla, "Evolving computer programs without subtree crossover," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 3, pp. 209–216, Sept. 1997.

[14] J. Schrum and R. Miikkulainen, "Constructing complex npc behavior via multi-objective neuroevolution," in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2008.