Interactive Evolution for the Procedural Generation of Tracks in a High-End Racing Game

Luigi Cardamone
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
cardamone@elet.polimi.it

Daniele Loiacono
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
loiacono@elet.polimi.it

Pier Luca Lanzi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
lanzi@elet.polimi.it

ABSTRACT

We present a framework for the procedural generation of tracks for a high-end car racing game (TORCS) using interactive evolution. The framework maintains multiple populations and allow users to work both on their own population (in single-user mode) or to collaborate with other users on a shared population. Our architecture comprises a web frontend and an evolutionary backend. The former manages the interaction with users (e.g., logs registered and anonymous users, collects evaluations, provides access to all the evolved populations) and maintains the database server that stores all the present/past populations. The latter runs all the tasks related to evolution (selection, recombination and mutation) and all the tasks related to the target racing game (e.g., the track generation). We performed two sets of experiments involving five human subjects to evolve racing tracks alone (in a single-user mode) or cooperatively. Our preliminary results on five human subjects show that, in all the experiments, there is an increase of users' satisfaction as the evolution proceeds. Users stated that they perceived improvements in the quality of the individuals between subsequent populations and that, at the end, the process produced interesting tracks.

Categories and Subject Descriptors

K.8 [**Personal Computing**]: Games

General Terms

Experimentation, Design

Keywords

Interactive Genetic Algorithms, Racing Games, Procedural Content Generation

1. INTRODUCTION

The automatic generation of game content has been a central issue for the game industry since the early 1980s, when the limitation of existing platforms did not allow the distribution of large amounts of pre-designed content (typically game levels). Accordingly, ad-hoc algorithmic procedures were widely applied to generate game content on-the-fly, so as to provide infinite levels of fun [13], and the field of *Procedural Content Generation* (or PCG) was born. Nowadays, this area has become crucial as most commercial games requires huge amounts of game content which would be unbearable for human designers (see for instance the infinite universes of *Eve Online*[1]).

In the recent years, several researchers have investigated the application of evolutionary methods for the automatic discovery of innovative and interesting game content. In this area, dubbed Search-Based Procedural Content Generation (SB-PCG) [25], the quality assessment of the evolved content is probably the most critical issue. So far, two approaches have been proposed in the literature (e.g., [22, 26]). *Theory-driven approaches* rely on domain experts who design ad-hoc heuristics (typically inspired to a theory of fun [12]) to measure the quality of game content; a limitation of these approaches is that such heuristics are typically difficult to implement and often biased toward the underlying theory. In contrast, *data-driven approaches* exploit huge collection of game data to build a predictive model to evaluate the quality of unseen content. These approaches have two major drawbacks in that (i) the data collection phase is typically time-consuming, as it may involve several players, and (ii) building an accurate model of players' satisfaction is usually challenging.

In this work, we take a completely different approach and investigate the application of interactive evolution for SB-PCG in the realm of car racing games. This is a rather popular game genre in which the content plays a key role for the commercial success of the title (see for instance, Trackmania by Nadeo[2] and rFactor by Image Space Inc.[3]). For this reason, the genre recently attracted the interest of researchers who applied methods of SB-PCG to evolve different types of content such as competitive driving behaviors [4], racing lines [5], and tracks [22, 15].

Our approach investigates the human-assisted generation of racing tracks for a high-end racing game (TORCS [1]);

[1] *http://www.eveonline.com/*
[2] http://www.trackmania.com
[3] http://www.rfactor.net

it is inspired to the early work of [22, 15], who applied SB-PCG to a simple 2D game [22] and to a high-end racing game [15] to evolve racing tracks which could either fit a target player profile [22] or provide a large degree of diversity [15]. Both [22, 15] assess the quality of game content using several statistics collected during one or more races involving non-player characters (or bots). In contrast, in this work, users define what they consider interesting game content and employ interactive genetic algorithms to search for the racing tracks they like most. For this purpose, we developed a framework which enables the interactive evolution of racing tracks both by single users, working alone on their own population, and by a community of users sharing (a typically much larger) population. Our framework comprises one web frontend and an evolutionary backend. The frontend is responsible for (i) managing the interaction with the users, (ii) rendering active populations, (iii) collecting users' evaluations, and (iv) maintaining the database of evolving/evolved populations. The evolutionary/gaming backend is in charge of (i) generating the content for the actual target game, (ii) performing all the evolutionary tasks (selection, recombination, and mutation) on all the currently active population. The partitioning between a web/database dedicated frontend and an evolutionary/gaming dedicated backend has been mainly introduced both (i) to improve the load-balancing between the generally light-weighted user interface related tasks and the CPU demanding evolutionary and game related tasks, but also (ii) to improve the framework portability (in fact, game executables usually require libraries that are not generally available on the most typical webservers). We validated our framework by performing two sets of experiments involving five human subjects, four different single-user setups and one cooperative multi-user setup. Our preliminary results show that, in all the experiments, there is an increase of users' satisfaction as the evolution proceeds. Users stated that they perceived improvements in the quality of the individuals between subsequent populations and that, at the end, the process produced interesting tracks.

2. SEARCH-BASED PROCEDURAL CONTENT GENERATION

Procedural Content Generation (PCG) dates back to the early 1980s when simple algorithmic procedures were applied to generate huge, possibly infinite, amount of game content with very limited resources. Some of the early examples in this area are represented by the games *Rogue*[4], *Elite*[5], and *MidWinter*[6]. Nowadays, although the memory limitations are long forgotten, procedural content generation is still widely used both to reduce the design costs and to generate those immense scenarios which would be infeasible for human designers. Recent examples include [2], *Diablo*, *Diablo II* in which maps were procedurally generated on-the-fly; and *Eve Online* which involves a universe generated using fractal methods.

Search-Based Procedural Content Generation (SB-PCG) extends PCG by replacing the handcrafted human design with search-based (usually evolutionary computation) methods [25]. Early examples in this area include the work of Hastings et al. [9] who applied a modified version of NEAT [20], NEAT Particles, to interactively evolve complex and interesting graphical effects to be embedded in computed games so as to enrich their content. The work was extended in [10] where the authors introduced the game Galactic Arms Race and provided the first demonstration of evolutionary content generation in an on-line multi-player game. Marks and Hom [16] were the first to evolve a set of game rules to obtain a balanced board game which could be equally hard to win from either side and rarely ended with a draw. Togelius et al. [22] combined procedural content generation principles with an evolutionary algorithm to evolve racing tracks for a simple 2D game which could fit a target player profile. Togelius and Schmidhuber [23] evolved complete rule sets for games. The game engine was capable of representing simple Pacman-style games, and the rule sets described what objects were in the game, how they moved, interacted (with each other or with the agent), scoring and timing. Frade et al. [7] applied Genetic Programming to the evolution of terrains for games which would attain the aesthetic feelings and desired features of the designer. The approach is currently employed in the game Chapas[7]. More recently, Togelius et al. [24] applied multi-objective evolution to evolve maps for *StarCraft* using a set of fitness functions evaluating the player's entertainment. Sorenson and Pasquier [19] presented a generative approach for level creation following a top-down approach and validated it using *Super Mario Bros.* and a 2D adventure game similar to the *Legend of Zelda*[8].

3. INTERACTIVE EVOLUTION FOR GAME CONTENT GENERATION

Interactive evolutionary algorithms measure the fitness of individuals through the interactions with users. Accordingly, these methods are typically applied to problems for which a computable fitness function is difficult or even impossible to define [21]. On the other hand, interaction with humans brings new challenges such as users' fatigue. For instance, Takagi [21] presents a review of the research efforts for abating fatigue in interactive evolutionary systems; Llorà et al. [14] introduced an addition synthetic fitness function to reduce the number of users' evaluations; Gong et al. [8] apply clustering to reduce the initial population size.

Interactive evolution has been applied to a wide range of domains including the generation of HTML styles [17]; fashion design [11]; ergonomic design [3]; the generation of terrains and landscapes [27], synthetic images [18], music [28], and art in general. To the best of our knowledge, Galactic Arm Race (GAR) [10] is the only application of interactive evolution to the generation of game content. Galactic Arms Race is a multiplayer online video game driven by methods of automatic content generation. It features a unique weapon systems that automatically evolves based on players' behavior through a specialized version of the NEAT evolutionary algorithm called cgNEAT (content-generating NeuroEvolution of Augmenting Topologies) [10].

[4]http://en.wikipedia.org/wiki/Rogue_(computer_game)
[5]http://en.wikipedia.org/wiki/Elite_(video_game)
[6]http://en.wikipedia.org/wiki/Midwinter_(video_game)

[7]https://forja.unex.es/projects/chapas
[8]Miyamoto, S., Nakago, T., Tezuka, T.: The Legend of Zelda. Nintendo (1986)

4. OUR FRAMEWORK

The structure of our interactive evolutionary system is depicted in Figure 1. The systems consists of two servers. The web frontend (i) manages the interaction with the users, (ii) renders active populations, (ii) collects evaluations of the users, (iii) maintains the database of evolving/evolved populations, and (iv) updates the request queues for the evolutionary backend. The evolutionary backend runs (i) all the evolutionary operators (selection, recombination, and mutation) on the currently active populations, (ii) the generation of actual game content (i.e., the mapping procedures between genotypes and phenotypes), and (iii) any other TORCS-related procedure such as the rendering of the track thumbnail pictures, which requires the invocation of a TORCS executable. The partitioning between a web/database dedicated frontend and an evolutionary dedicated backend was mainly introduced to improve the load-balancing between the generally light-weighted user interface related tasks and the CPU demanding evolutionary and TORCS related tasks. In addition, since TORCS executables require libraries that are not generally available on the most typical webservers, we decoupled the the web-related application from the TORCS-related applications also to make our system portable to more hosting services.

4.1 The Frontend User Interface

People can access the framework using a browser by logging in either as registered or anonymous users, and can request either to start/join a single-user session or to join an active multi-user session.

Registered users are fully tracked and thus can (i) save the state of an active evolutionary process, (ii) continue a previously saved process, and (iii) explore all their history (e.g., all the populations they evolved, all their preferences). In contrast, anonymous users are only tracked using browsers' cookies so that the status of their evolutionary process is lost as soon as the browser cookies are cleared or they connect from a different machine.

In single-user mode, users have the complete control over the evolutionary process in that (i) they can customize the parameter set up by specifying the population size, the selection policy (tournament or truncation), etc. (ii) they can evaluate *all* the individuals in the population, and (iii) they can request the next selection, recombination, mutation cycle. As an example, Figure 2 shows the interface for a single-user evolution. The upper section of the interface includes all the commands available to the user: `Evolve` requests the next selection, recombination and mutation step to the evolutionary backend and then updates the current page with the new population; `Reset` restarts the evolution from scratch (all the previous populations are stored and a new random population is generated); `Generations` provides access to the data of all the previous generations.

The lower section of the interface depicts the current population and, for each track, it shows (i) the track thumbnail and (ii) its scoring interface. In particular, the framework provides two scoring interfaces (Figure 3): a ranking interface (see Figure 2 and Figure 3a) which asks users to rank an individual with an integer between 1 and 5; a like/dislike interface (Figure 3b), which asks users simply whether they like the track.

In multi-user mode, users have very limited control over the process in that they are *only* allowed (i) to explore and evaluate a small number individuals, randomly selected from the much larger underlying population, and (ii) to access the hall of fame of the best individuals evolved so far. In this mode, users cannot specify the parameters of the genetic algorithm, nor they can request the activation of the evolutionary cycle which is actually triggered by a heuristic based on the number of evaluations received.

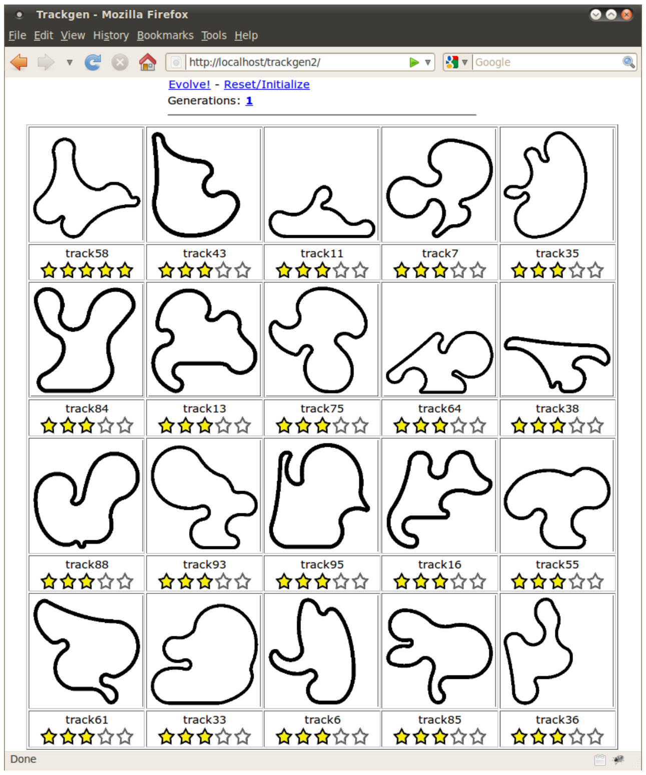

Figure 2: A screenshot of the User Interface.

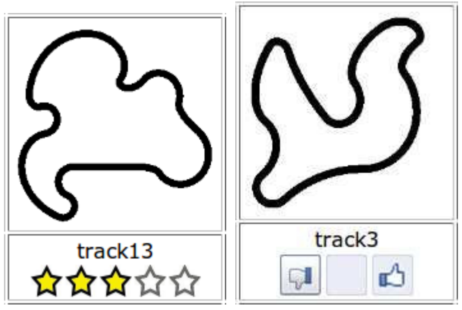

Figure 3: Scoring interfaces: (a) the ranking interface asks users to rank an individual with an integer between 1 and 5; (b) the like/dislike asks users to specify just whether they like the track.

4.2 The Evolutionary Backend

The backend runs of all the evolutionary and TORCS related tasks. It hosts a number of servers (labeled GA in Figure 1) each one listening to a separate request queue. GA servers are mainly responsible of (i) running the selection, recombination, mutation cycle; (ii) rendering the newly created populations, which requires the execution of TORCS

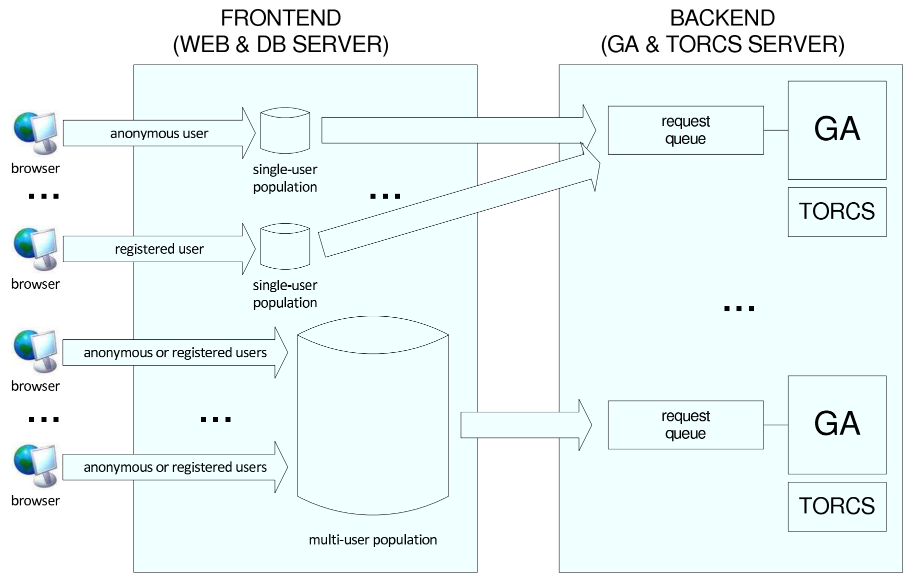

Figure 1: Structure of the interactive evolutionary system.

applications; and (iii) updating the status of the evolutionary process on the main databases. A selection, recombination, mutation cycle can be requested either (i) explicitly, in single-user mode, by the user through the `Evolve` command on the main interface, (ii) implicitly, in multi-user mode, by the GA server, when a sufficient number of evaluations for the individuals in the population has been received. Note that, for load balancing purposes, all the incoming requests from the web frontend are queued so that each GA server can explicitly manage the number of running processes. When a new generation is required, a GA server first performs the standard selection, recombination, and mutation; then, for each one of the newly created individuals, the server runs a TORCS executable that generates the actual TORCS track (the phenotype), computes several track statistics, and render the track shape; finally the status on the database is updated. Since TORCS executables tend to be CPU and disk intensive, the GA server schedules the number of active TORCS invocations based on the number of cores available to avoid overloading the server.

GA servers implement a rather simple real-coded genetic algorithm that accesses a population of tracks stored on a remote database, using tournament or truncation selection, single-point crossover, and mutation. When an infeasible track is generated, it is discarded and another one is generated. In single-user mode, when very small population are involved, the 10% of a new population is filled with randomly generated tracks so as to avoid premature convergence.

5. TORCS

The Open Racing Car Simulator (TORCS) [1] is a state-of-the-art open-source car racing simulator which provides a sophisticated physics engine, full 3D visualization, several tracks, car models, and game modes (practice, quick race, championship, etc.). The car dynamics is accurately simulated and the physics engine takes into account many aspects of racing cars such as traction, aerodynamics, fuel consumption, etc. All the experiments reported in this paper have been carried out with TORCS 1.3.1.

6. TRACK REPRESENTATION FOR CONTENT GENERATION

The encoding of game content is a central issue for Search-Based Procedural Content Generation [25]. In this section, we briefly describe the TORCS representation of tracks (*the phenotype*), the encoding we designed (*the genotype*), and the procedures to map genotypes into phenotypes.

The Phenotype. A track in TORCS is represented as an ordered list of segments. Each segment is either a straight or a turn. A straight is defined by just one parameter, its length. A turn is defined by (i) the direction (i.e., left or right); (ii) the arc it covers measured in radians; (iii) its start radius and (iv) its end radius. In addition, the track must be feasible, i.e., it must be *closed*, and therefore the last segment must overlap the first segment.

The Genotype. The direct encoding of the track representation in TORCS into a genotype is infeasible as it would result in a *huge* search space (thus leading to the curse of dimensionality) containing only a tiny fraction of feasible (closed) tracks. Accordingly, we used an *indirect encoding* inspired to the work of Togelius et al. [22] on a simple 2D car racing simulator. In [22], a track is represented as a set of *control points* that the track has to cover; the track is generated as a sequence of Bezier curves connecting three control points and, to guarantee smoothness, have the same first and second derivatives at the point they join. In this work, we encoded a track as a sequence of control points $\vec{p} = \{p_1, \dots p_n\}$, each one consisting of three parameters r_i, θ_i, and s_i; r_i and θ_i identify the position of the control point p_i in a polar coordinate system (r_i is the *radial coordinate*, θ_i is the *angular coordinate*); s_i controls the slope of the track tangent line in p_i. Figure 4 shows an example of our encoding: control points are depicted in red; the blue dot represents the origin of the polar coordinate system; the curves represent what generated by the genotype to phenotype mapping process, discussed in the next section.

Genotype to Phenotype Mapping. This procedure takes as input the n control points $\vec{p}$ and returns a list $\vec{t}$ of track segments in TORCS format. Initially, the polar coordinates of the n control points (i.e., the r_i and θ_i values) are used to compute the ranges of feasible slope values for each control point. If there exist a control point for which there are no feasible slope values (thus it is not possible to joint the incoming and the outgoing segments in that point), no track can be derived from $\vec{p}$ and therefore no track (a NULL track) is returned. Otherwise, given the ranges of feasible slope values, the actual slope values are computed on the basis of the s_i values. Next, for each pair of control points, p_i and p_{i+1}, the corresponding TORCS segment is generated using both the position and the slope values as constraints. Then, the procedure tries to close the track by generating one or more segments to connect t_{n-1} to t_1. If the process succeeded, the resulting track $\vec{t}$ is returned otherwise a NULL track is returned.

Generating Closed Tracks. In general, it is not always possible to connect the last and first control points with only one segment and a sequence of straights and turns might be needed. Accordingly, a procedure CLOSETRACK considers the parameters defining the end and starting control points p_n and p_1 and try different heuristics to generate a feasible closed track.

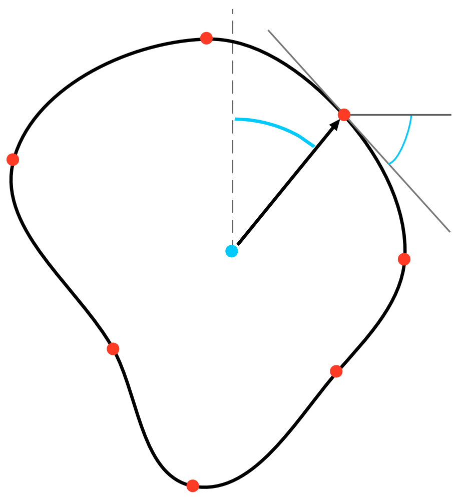

Figure 4: Track encoded as a set of control points(the red dots) represented in a polar coordinate system; the blue dot is the system origin; the solid line shows an example of a feasible slope generated by the genotype to phenotype mapping.

7. EXPERIMENTAL VALIDATION

We performed two sets of experiments involving five users to provide an initial validation of the proposed framework. In the first set of experiments, we focused on the single-user mode and investigated how (i) the scoring interface (i.e., either a ranking interface or a like/dislike interface) and (ii) the selection policy (i.e., either tournament or truncation selection) affect the quality of the evolutionary process. In the second set of experiments, we studied the system working in multi-user mode.

7.1 Single-User Mode

In the first set of experiments, we tested four different configurations: (i) the ranking scoring interface combined with tournament selection; (ii) the like/dislike scoring interface combined with tournament selection; (iii) the ranking scoring interface combined with truncation selection policy; (iv) the like/dislike scoring interface combined with truncation selection policy. For each configuration, we asked five human subjects to complete ten generations of single-user evolution using a population consisting of 20 individuals. During the experiments subjects did not know the type of selection mechanisms in use (tournament or truncation). Since subjects could access the system as registered users, they were allowed to pause and restart the evolutionary process as they wished (which, in principle, should have limited their fatigue). However, our tests show that the evaluation of a population with 20 individuals for 10 generations would typically take around 20-30 minutes of actual time (i.e., with no pause in between evaluations). To measure the progress during the evaluation process, for each generation we computed the average score of the individuals in the population as follows. When the ranking interface is used, a integer value (i.e., a fitness) between 1 and 5 is assigned to each track according to the user preferences. When the like/dislike interface is used, a (fitness) value of 1 is assigned to tracks the user does not like whereas a (fitness) value of 5 is assigned to tracks the user likes.

We first analyzed how the scoring method influences the evolutionary process. For this purpose, we grouped all the collected data based on the scoring method used. Figure 5 compares the average population scores for the runs using the ranking interface (empty dots) and the like/dislike interface (solid dots). Both curves show an improvement in the users satisfaction as the number of generation proceeds. The use of the simpler like/dislike interface appears to provide a more evident increase of the user satisfaction, whereas the ranking interface results in a slight improvement over time. This result was later confirmed by the feedback we received by the subjects who stated they perceived the simpler (like/dislike) interface as more effective to express their preferences while they found it difficult to provide meaningful/coherent overall rankings.

We performed a similar analysis to study how selection may influence the evolutionary process. Accordingly, we grouped all the collected data based on the selection method; it is important to stress that, users did not know the selection method used during the experiments. Figure 6 compares the average population scores for the runs using tournament (empty dots) and truncation selection (solid dots). Again both curves show an improvement in the users satisfaction as the number of generation proceeds. In this case however, there is no noticeable difference between the trials using truncation or tournament selection, suggesting that the selection mechanism had no influence on the users.

Overall, the feedback we received from the subjects was generally positive. In most cases, users reported that they perceived improvements in the quality of the individuals between subsequent populations. They also found that, at the end, the process produced interesting tracks. As an example, Figure 8 and Figure 9 show two of the most interesting tracks evolved according to the users. The only criticism we recorded concerned the ranked scheme which appears to cause, sometimes, fatigue and frustration in the users. In

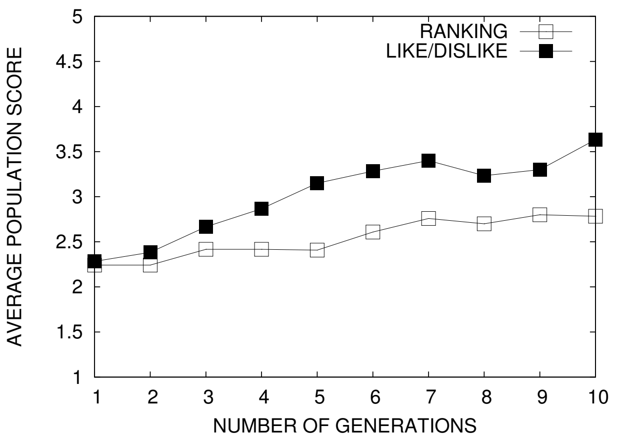

Figure 5: Average population score for the trials using the like/dislike evaluation (solid dots) and the ranked evaluation (empty dots). Curves are averages over 10 trials.

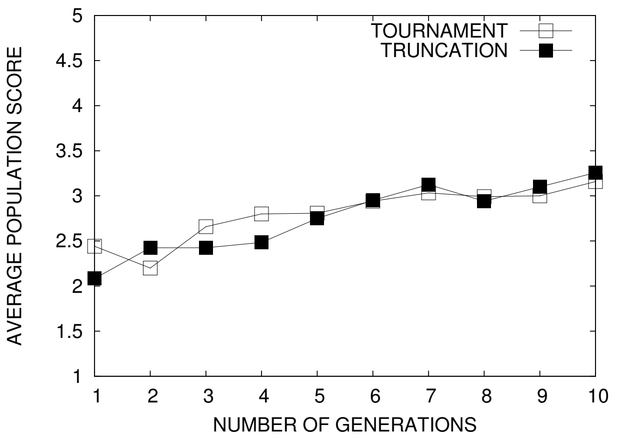

Figure 6: Average population score for the trials using truncation selection (solid dots) and tournament selection (empty dots). Curves are averages over 10 trials.

addition, users told us they would tend to be annoyed when many very similar individuals appeared in the same population.

7.2 Multi-User Mode

At the end, we performed a preliminary experiment to test the multi-user mode. The experiment involved the same five human subjects, participating in the previous experiments, who in this case shared the same population; subjects could not communicate one another in anyway while the experiment was running. Given the limited number of subjects involved, we used a population of only 20 individuals (the same size used in the previous experiments) and thus allowed all the subject to score all the individuals. Tracks were evaluated using the simpler like/dislike interface; track fitness was computed as the average score received from the users; truncation selection was applied.

Figure 7 reports the average population score over the number of generations. As in the single-user experiments,

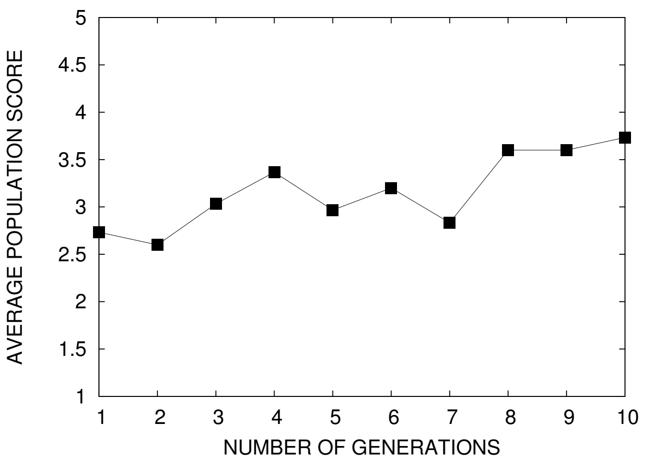

Figure 7: Average population score for one trial using like/dislike scoring and truncation selection.

the curve shows a clear improvement in the users' satisfaction, in fact, the number of likes increases as the evolution proceeds, notwithstanding possible conflicting opinions that the users have expressed.

8. CONCLUSIONS

We introduced a framework which exploits interactive evolution for the human-assisted generation of tracks for a high-end open-source car racing game (TORCS). The framework comprises a web frontend which manages the interactions with anonymous/registered users who can work on their own population (in single-user mode) or can cooperate on a shared population (in multi-user mode). An evolutionary backend manages both all the instances of interactive genetic algorithms and all the tasks strictly connected to the target racing game (e.g., the generation of the actual in-game content, the rendering of the thumbnails needed by the web frontend). We validated the initial prototype of the framework with five human subjects who were asked to perform four single-user trials and one multi-user trial involving all the subjects together. The preliminary feedback we received was generally positive. In most cases, users perceived improvements in the quality of the tracks between subsequent generations. Users also stated that, at the end, the process produced some interesting tracks. The only criticism we recorded concerned the evaluation interface which requires to rank all the tracks. In fact, this seems to cause fatigue and frustration in the users who did not feel comfortable in providing a complete ranking of all the tracks. Users also told us that they tended to be annoyed, in later generations, when many very similar individuals appeared in the same population.

Our framework is still at a preliminary stage but the feedback we received, from the users who tried it, is very promising. At the moment, we are working on new features, which the feedback we received suggested, that will also require further validation with human subjects. First, we plan to improve the rendering of populations by introducing a clustering preprocessing to group together very similar track into one single representative. This should alleviate the frustration we recorded when more copies of very similar tracks are

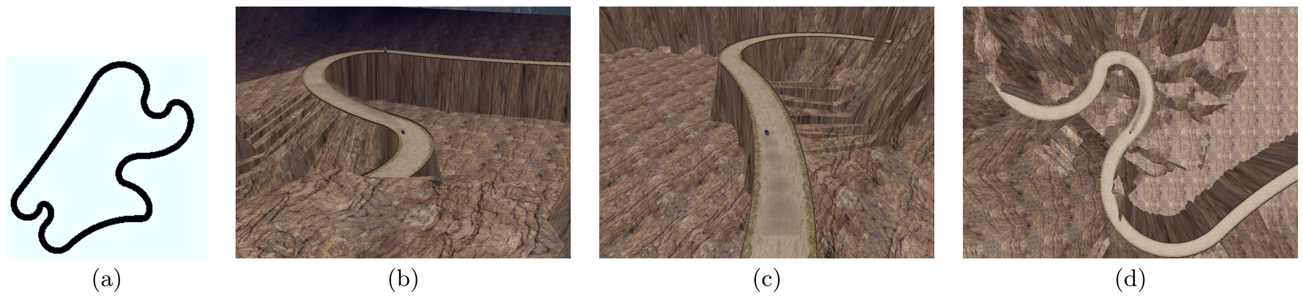

Figure 8: First example of track evolved in single-user mode: (a) shape of the track as used during the evolution; (b), (c), and (d) actual in-game renderings.

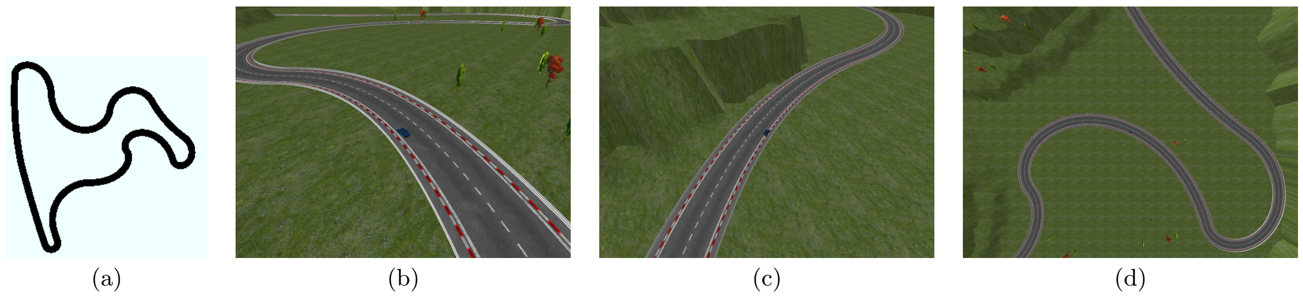

Figure 9: Second example of track evolved in single-user mode: (a) shape of the track as used during the evolution; (b), (c), and (d) actual in-game renderings.

shown. Second, we plan to add a sort of zooming operator which would apply a local search to some selected individuals the user particularly likes. Finally, we plan to add niching mechanisms to multi-user evolution so as to maintain higher degrees of diversity as the number of users and the population size increases.

The next big step will be to make the framework publicly available as a service for the TORCS community and also for the communities of other related games such as Speed Dreams[9] and VDrift[10]. Our long term goal is to create an on-line community which would make these open-source games more attractive by enabling the generation of large quantities of high-quality content. In our opinion, in fact, the limited availability of game content is currently one of the major limitations of most open-source games.

9. REFERENCES

[1] The open racing car simulator website. http://torcs.sourceforge.net/.

[2] Procedural content generation. http://pcg.wikidot.com/.

[3] A.M. Brintrup, J. Ramsden, H. Takagi, and A. Tiwari. Ergonomic chair design by fusing qualitative and quantitative criteria using interactive genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 12(3):343 –354, 2008.

[4] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.

[5] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Racing line optimization using genetic algoritms. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 388–394, Copenhagen, Denmark, August 18-21 2010. IEEE.

[6] Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna Esparcia-Alcázar, Chi Keong Goh, Juan J. Merelo Guervós, Ferrante Neri, Mike Preuss, Julian Togelius, and Georgios N. Yannakakis, editors. *Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, volume 6024 of *Lecture Notes in Computer Science*. Springer, 2010.

[7] Miguel Frade, F. Fernandez de Vega, and Carlos Cotta. Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience. In Mario Giacobini et al., editor, *Applications of Evolutionary Computing*, volume 4974 of *LNCS*, pages 485–490, Napoli, Italy, 2008. Springer.

[8] Dunwei Gong, Jie Yuan, and Xiaoping Ma. Interactive genetic algorithms with large population size. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1678 –1685, 2008.

[9] *http://www.speed-dreams.org*
[10] *http://vdrift.net*

[9] E. Hastings, R. Guha, and K.O. Stanley. Neat particles: Design, representation, and animation of particle system effects. In *Proc. IEEE Symposium on Computational Intelligence and Games CIG 2007*, pages 154–160, 2007.

[10] Erin J. Hastings, Ratan K. Guha, , and Kenneth O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):245–263, 2009.

[11] Hee-Su Kim and Sung-Bae Cho. Application of interactive genetic algorithm to fashion design. *Engineering Applications of Artificial Intelligence*, 13(6):635 – 644, 2000.

[12] Raph Koster. *Theory of Fun for Game Design*. PARAGLYPH PRESS, 2005.

[13] John E. Laird and Jonathan Schaeffer, editors. *Procedural Level Design for Platform Games*. The AAAI Press, 2006.

[14] Xavier Llorà, Kumara Sastry, David E. Goldberg, Abhimanyu Gupta, and Lalitha Lakshmi. Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1363–1370, New York, NY, USA, 2005. ACM.

[15] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Automatic track generation for high-end racing games using evolutionary computation. Technical Report 2011.08, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2011.

[16] Joe Marks and Vincent Hom. Automatic design of balanced board games. In Jonathan Schaeffer and Michael Mateas, editors, *AIIDE*, pages 25–30. The AAAI Press, 2007.

[17] N. Monmarche, G. Nocent, M. Slimane, G. Venturini, and P. Santini. Imagine: a tool for generating html style sheets with an interactive genetic algorithm based on genes frequencies. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 3, pages 640 –645 vol.3, 1999.

[18] Jimmy Secretan and Nicholas Beato. Picbreeder: evolving pictures collaboratively online. In *CHI*, pages 1759–1768, 2008.

[19] Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. In Chio et al. [6], pages 131–140.

[20] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural network through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[21] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275 –1296, September 2001.

[22] J. Togelius, R. De Nardi, and S.M. Lucas. Towards automatic personalised content creation for racing games. In *Proc. IEEE Symposium on Computational Intelligence and Games CIG 2007*, pages 252–259, 2007.

[23] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 111–118, Dec. 2008.

[24] Julian Togelius, Mike Preuss, Nicola Beume, Johan Hagelbaeck Simon Wessing, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 265–262, Copenhagen, Denmark, 18-21 August 2010., 2010. IEEE.

[25] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation. In Chio et al. [6], pages 141–150.

[26] Niels van Hoorn, Julian Togelius, Daan Wierstra, and Jürgen Schmidhuber. Robust player imitation using multiobjective evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 652–659, Trondheim, Norway, 18-21 May 2009.

[27] P. Walsh and P. Gade. Terrain generation using an interactive genetic algorithm. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –7, 2010.

[28] Bin Xu, Shangfei Wang, and Xian Li. An emotional harmony generation system. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –7, 2010.