Applied Soft Computing 28 (2015) 550–558

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

TrackGen: An interactive track generator for TORCS and Speed-Dreams

Luigi Cardamone, Pier Luca Lanzi*, Daniele Loiacono

Dipartimento di Elettronica, Informazione e Bioinformatica, Politecnico di Milano, Milano, Italy

ARTICLE INFO

Article history:
Received 31 July 2013
Received in revised form
12 November 2014
Accepted 12 November 2014
Available online 15 December 2014

Keywords:
Evolutionary computation
Video games
Interactive evolution
Procedural content generation

ABSTRACT

TrackGen is an online tool for the generation of tracks for two open-source 3D car racing games (TORCS and Speed Dreams). It integrates interactive evolution with procedural content generation and comprises two components: (i) a web frontend that maintains the database of all the evolved populations and manages the interaction with users (by collecting users evaluations and providing access to the evolved tracks) and (ii) an evolutionary/content-generation backend that runs both the evolutionary algorithm and generates the actual game content that is available through the web frontend. The first prototype of the tool was presented in July 2011 but advertised only to researchers; the first official version which generated tracks only for TORCS was released to the game community in September 2011; due to the many requests, we released a new version soon afterwards, in January 2012, with support for Speed Dreams (the fork of TORCS focused on visual realism and graphic quality) that has been online since then. From January 2012 until July 2014, TrackGen had more than 7600 unique visitors who visited the website around 11,500 times and viewed 85,500 pages; it was employed to evolve more than 8853 tracks, and it was used to download 12,218 tracks. Some of the tracks evolve by our system have been also included in the TORCS distribution.

1. Introduction

Procedural Content Generation (PCG) dates back to the early 1980s when it was introduced to overcome the memory limitations of existing platforms and employed to distribute large amounts of pre-designed game content. After several decades, although the limitations that originated it have long gone, procedural content generation maintains its crucial role in the game industry as a mean to design, produce, and deliver huge amounts of game content, that would be unbearable for human designers (e.g., the infinite universes of *Eve Online*[1]). At the same time, procedural content generation has now become an advertised feature of commercial games as in the *Borderlands* franchise,[2] *Spore*,[3] *Tiny Wings*,[4] and many others [1].

Search-Based Procedural Content Generation (SB-PCG) [2] is a branch of PCG that applies stochastic search algorithms to generate high-quality game content (e.g., levels, rules, weapons). In this paper, we present TrackGen, a tool that applies SB-PCG, more precisely interactive evolution, to the realm of car racing games, a popular genre in which the content plays a key role for the commercial success of the title (see for instance, *Trackmania*[5] and *rFactor*[6]).

TrackGen is an online SB-PCG tool that evolves tracks for two open-source 3D car racing games (TORCS [3] and Speed Dreams [4]). TrackGen is inspired by the early work of Togelius et al. [5,6], who applied SB-PCG to a simple 2D game to evolve racing tracks which could fit a target player profile [5,6]. It is also inspired by our previous work on the automatic evolution of TORCS tracks [7] which could provide a large degree of diversity. Both [5–7] evaluate the quality of game content using several statistics collected during one or more races involving non-player characters. In contrast, TrackGen let users state what they consider interesting game content and employs an interactive genetic algorithm to search for the better tracks; in this respect, TrackGen has been strongly inspired by the work on *Galactic Arms Race*[7] [8].

TrackGen was first shown in its prototypical form during CIG-2011 and GECCO-2011, when a seminal paper about the tool was presented [9]. The first official version was released later in September 2011 to the TORCS community and received much feedback. In particular, we were asked to extend the tool to Speed Dreams [4], a fork of TORCS focused on visual realism and high-quality graphics. The second version that could produce tracks both for TORCS and Speed Dreams was released in January 2012 and has been online since then. The new version is also connected to a Twitter account (*http://twitter.com/POLIMIVGR*) and tweets an announcement every time new tracks are generated. Four tutorial videos available at the TrackGen website[8] and YouTube demonstrate TrackGen main features and show examples of the high-quality tracks that the system is able to generate. So far, TrackGen had more than 7600 unique visitors who visited the website around 11,500 times and viewed 85,500 pages; it was employed to evolve more than 8853 tracks, and it was used to download 12,218 tracks. Some of the tracks evolve by our system have been also included in the TORCS distribution in an additional package of extra tracks.

* Corresponding author. Tel.: +39 0223993472; fax: +39 0223993411.
E-mail address: pierluca.lanzi@polimi.it (P.L. Lanzi).

[1] http://www.eveonline.com/.
[2] http://www.borderlandsthegame.com/ and http://www.borderlands2.com/.
[3] http://www.spore.com/.
[4] http://www.andreasilliger.com/.
[5] http://www.trackmania.com.
[6] http://www.rfactor.net.
[7] http://gar.eecs.ucf.edu/.
[8] TrackGen tutorial videos: http://trackgen.pierlucalanzi.net/about.htm; http://www.youtube.com/watch?v=YFOa7L3oBwM; http://www.youtube.com/watch?v=0_W4jHN2h2Q; http://www.youtube.com/watch?v=3AzjMtRDnBo; http://www.youtube.com/watch?v=Si_u_43HJnM.

http://dx.doi.org/10.1016/j.asoc.2014.11.010
1568-4946/

2. Related work

Procedural Content Generation (PCG) dates back to the early 1980s when simple algorithmic procedures were applied to generate huge, potentially unlimited, amount of game content with very limited resources. Some of the early examples in this area are represented by the games *Rogue*,[9] *Elite*,[10] and *MidWinter*.[11] Nowadays, although the memory limitations are long overcome, 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 [1], the *Diablo* series in which maps are procedurally generated; *Spore* makes heavy use of PCG for animations, *FarCry2* uses PCG for vegetation; *Borderland* and *Borderland2* use PCG for weapons; and *Eve Online* involves a universe generated using fractal methods.

2.1. Search-Based Procedural Content Generation (SB-PCG)

SB-PCG extends traditional procedural content generation by replacing the handcrafted human design with search-based (usually evolutionary computation) methods [2]. Early examples in this area include the work of Hastings et al. [10] who applied a modified version of NEAT [11], named 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 [8] where the authors introduced the game *Galactic Arms Race* and provided the first demonstration of evolutionary content generation in an online multi-player game.

Marks and Hom [12] 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 and Schmidhuber [13] evolved complete rule sets for games. The game engine was capable of representing simple *Pac-Man* like 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. Later, Browne [14] developed the Ludo system to evolve rules for abstract connect games and published one of the evolved games, namely *Yavalath*.[12]

Togelius et al. [5,6] 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. Avery et al. [15] developed a tower defense game using experience driven procedural content generation. Cardamone et al. [16] evolved maps for the *Cube2*[13] first-person shooter that maximizes fighting time.

Frade et al. [17] applied Genetic Programming to the evolution of terrains for games which would attain the esthetic feelings and desired features of the designer. The approach is currently employed in the game *Chapas*.[14]. More recently, Togelius et al. [18] applied multi-objective evolution to evolve maps for *StarCraft* using a set of fitness functions evaluating the player's entertainment. The work was later extended in [19] where also an imaginary generic strategy game based on height maps (a rather common representation) was considered. Raffe et al. [20,21] presented an evolutionary algorithm for the *in-game generation* of terrains based on a set of height-map patches extracted from sample maps whose layout was optimized through evolution. The same authors also developed a third-person action game in which levels are tailored to the players' preferences and skill.[15]

Sorenson and Pasquier [22] 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*.[16] Shaker et al. [23] applied Grammatical Evolution to generate levels for *Mario Bros*. Friberger et al. [24] introduced the concept of *data games* as a way to make sense of the data through a game and applied PCG to evolve *Monopoly* boards based from economic and social indicator data for local governments in the UK.

Jordan et al. [25] developed a mobile game in which music (selected from a pool of songs) was analyzed and its features were used to guide the procedural generation of levels. Plans and Morelli [26] outlined the use of experience driven procedural music generation using user gameplay metrics.

2.2. Interactive evolution

Interactive evolution has been applied to a wide range of domains including the generation of HTML styles [27]; fashion design [28]; generation of facial composites [29]; semi-supervised learning [30]; voice quality conversion [31]; ergonomic design [32]; the generation of terrains and landscapes [33], synthetic images [34], music [35], and art in general. To the best of our knowledge, *Galactic Arm Race* [8] is the first application of interactive evolution for 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) [8]. Risi et al. [36] developed the first Facebook game, *Petalz*,[17] that applies interactive evolution to breed procedurally generated flowers.

3. TORCS and Speed Dreams

The Open Racing Car Simulator (TORCS) [3] 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, and fuel consumption.

Speed Dreams [4] is a fork of TORCS focused on visual and physics realism. While its core consists in a large portion of the original TORCS code base, Speed Dreams aims at providing better playing experience by introducing several features including more and visually improved car models and tracks, experimental physics engine, better AI bots, better menus, online multiplayer, great visual effects, etc.

4. Track representation

The encoding of game content is a central issue for Search-Based Procedural Content Generation [2]. In this section, we briefly illustrate the representation of tracks in TORCS/Speed Dreams, and the encoding used in TrackGen.

4.1. Track representation in TORCS/Speed Dream

TORCS and Speed Dreams share a large part of the code base. In particular, Speed Dreams uses an extended version of the track

[9] http://en.wikipedia.org/wiki/Rogue_(computer_game).
[10] http://en.wikipedia.org/wiki/Elite_(video_game).
[11] http://en.wikipedia.org/wiki/Midwinter_(video_game).
[12] http://www.cameronius.com/games/yavalath/.
[13] http://sauerbraten.org/.
[14] https://forja.unex.es/projects/chapas.
[15] http://goanna.cs.rmit.edu.au/ wraffe/ExperimentHome.html.
[16] https://en.wikipedia.org/wiki/The_Legend_of_Zelda.
[17] http://petalzgame.com/.

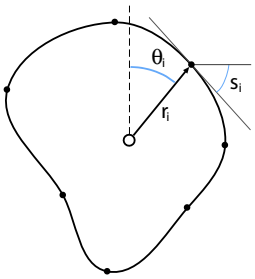

Fig. 1. Track encoded as a set of control points (the solid dots) that are represented in a polar coordinate system; the central empty dot is the system origin; the solid line shows an example of a feasible slope generated by the genotype to phenotype mapping.

representation in TORCS with additional features that improve the rendering and the special effects. Tracks are represented as ordered list of segments. Each segment is either a straight or a turn: a straight is defined just by 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.

4.2. Track representation in TrackGen

The direct encoding of a track representation 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 [5]. Accordingly, TrackGen applies the same *indirect encoding* we employed for the automatic evolution of TORCS tracks in [7], which was inspired by the work of Togelius et al. [5]. In TrackGen, a track is encoded as a sequence of control points $\vec{p} = \{p_1, \ldots, p_n\}$, 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. Fig. 1 shows an example of our encoding: control points are depicted as solid dots; the central empty dot represents the origin of the polar coordinate system; the curve that connects the solid dots is generated from the genotype (i.e.,) what generated by the genotype to phenotype mapping process, discussed in the next section.

Given a sequence of control points $\vec{p}$, a list of track segments in the TORCS/Speed Dreams format can be generated using the same procedure we used in [7]. At first, a range of feasible slopes for each control point p_i is generated; if none exists, the track is eliminated from the population; next, the actual slopes are generated from the feasible ones using a recursive procedure; finally, a specific algorithm is applied to compute the last slope that must close the track (see [7] for details).

5. TrackGen

Our tool comprises two main components hosted on two separate servers (Fig. 2). The frontend hosts the webpages, manages the database containing the evolved populations, collects user evaluations, maintains the request queue for the evolutionary backend, and finally sends the requests for downloadable content to the track and scenery generator. The backend runs both the evolutionary algorithm and all the tasks related to the generation of the actual game content including the mapping procedures between genotypes and phenotypes, the rendering of the track thumbnails, the generation of the downloadable tracks that are uploaded to the web frontend. This partition between a web/database dedicated frontend and an evolutionary/content-generation dedicated backend was mainly introduced to decouple the web applications from the TORCS/Speed Dreams-specific applications so to improve the system portability to other hosting services. In fact, the content-generation tasks require several TORCS and Speed Dreams executables that use libraries (e.g., SDL[18]) typically not available from most web hosting services.

5.1. The frontend user interface

TrackGen can be accessed using any cookie-enabled browser at http://trackgen.pierlucalanzi.net. The homepage (Fig. 3) shows a random selection of ten tracks from the underlying evolving population; for each track, the system depicts a thumbnail of the track shape and the track *unique name* selected from a public database containing the names of 100,000 places from all over the planet; if the page is refreshed another set of tracks is shown. Users can express their opinion about the track shape by pressing either the *thumb up* button to state that they like the track shape or the *thumb down* button to state that they dislike the track shape; otherwise, they can remain neutral by not pressing any of the above. TrackGen exploits browser cookies to track users that previously visited the page and remembers their previous evaluations; users can thus change their evaluation deciding for instance to dislike a track they previously liked or to remain neural by pressing the button between the two thumbs to clear a previous evaluation. By pressing the green arrow below a track thumbnail, users are sent to the download page (Fig. 4) where they can request the download of the track rendered using one of five possible sceneries: *city*, *desert*, *hill*, *mountain*, and *snow*.[19] All the download requests are queued to the track and scenery generator hosted on the backend that will build the tracks for TORCS and Speed Dreams using methods of procedural content generation and proper assets (see Section 5.3). When a download request is queued, a countdown based on the estimated preparation time is started; when the track has been uploaded to the frontend, a download link appears. Note that, all the previously generated tracks are stored on the frontend server, so that if the requested track is already available for download (because another user requested the same track before), the download link will immediately appear. Two tutorial videos, available at the TrackGen website[20] and YouTube, show the frontend user interface in action.

The TrackGen homepage has also a link to the *Hall of Fame* showing the ten tracks that received the highest score and a link to the *Latest* page showing the ten most recently evolved tracks. The latest evolved tracks are also advertised on the Twitter page of our research group, *http://twitter.com/POLIMIVGR*, where a tweet appears every time a new track has been generated by the evolutionary process.

5.2. The evolutionary algorithm

The evolutionary algorithm behind TrackGen is a simple steady-state real-coded genetic algorithm that runs on the backend as a *listener process* that is always active. When the evolutionary algorithm is initialized or restarted from scratch (Algorithm 1) all the tables on the web fronted are cleared (line 2), a new random population is generated and loaded on the web frontend (line 3),

[18] *http://www.libsdl.org/*.
[19] A video showing the five scenarios is available at http://www.youtube.com/watch?v=Si_u_43HJnM.
[20] TrackGen tutorial videos: http://trackgen.pierlucalanzi.net/about.htm; http://www.youtube.com/watch?v=YFOa7L3oBwM; http://www.youtube.com/watch?v=0_W4jHN2h2Q.

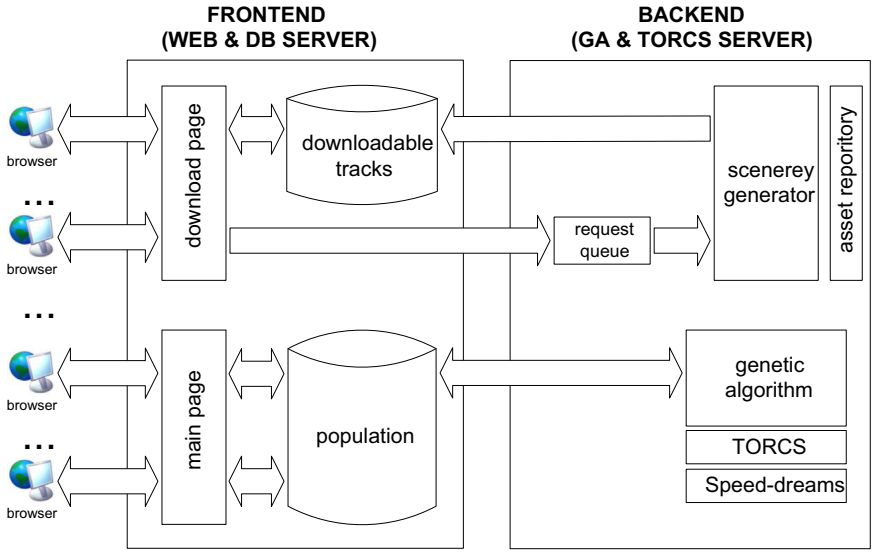

Fig. 2. Architecture of TrackGen.

and the tables containing all the statistics about the population and the process are initialized (line 4).

The evolutionary process (Algorithm 2) works as follows. At fixed time intervals, the process query the database on the web frontend to check the number of new evaluations received for the current population (line 3); if a sufficient number of new evaluations since the last activation of the evolutionary algorithm has been received, the process runs an evolutionary cycle (lines 6 to 18). At first, the current population is retrieved from the database on the web frontend (line 6) as a table containing the individuals (represented as a vector of real values) and their current fitness values; the fitness of a track is computed as the difference between

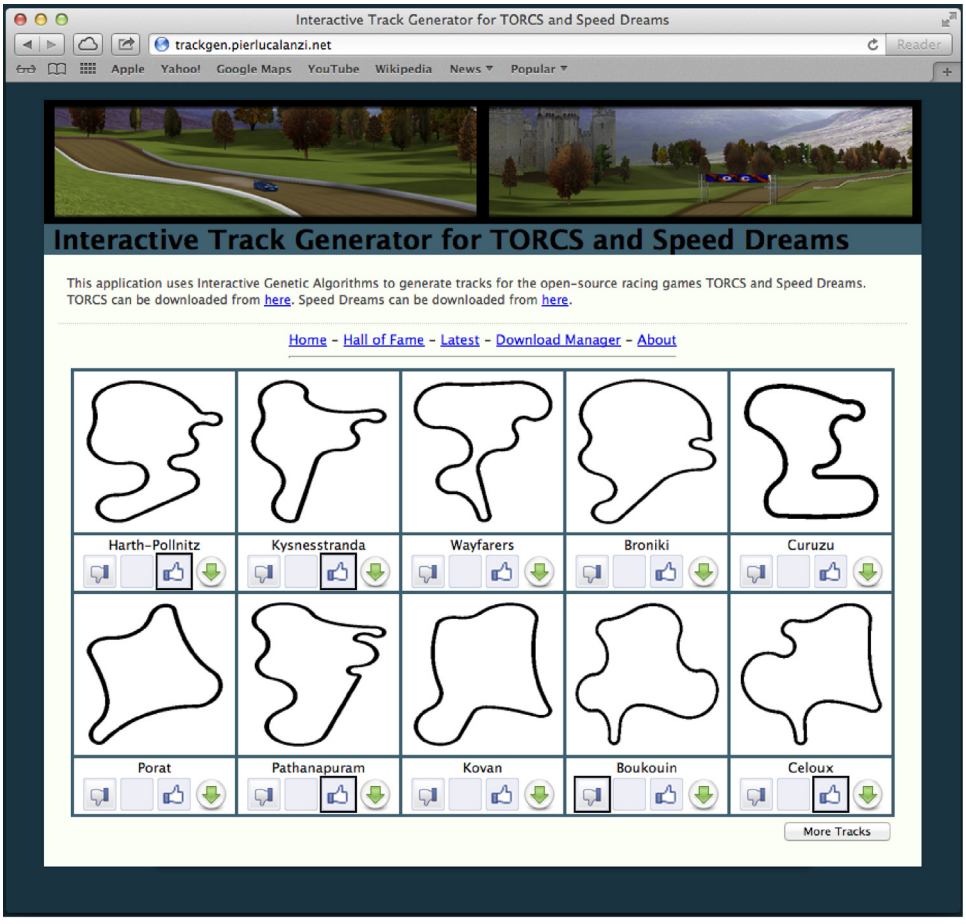

Fig. 3. The TrackGen homepage displaying ten tracks randomly selected from the underlying population.

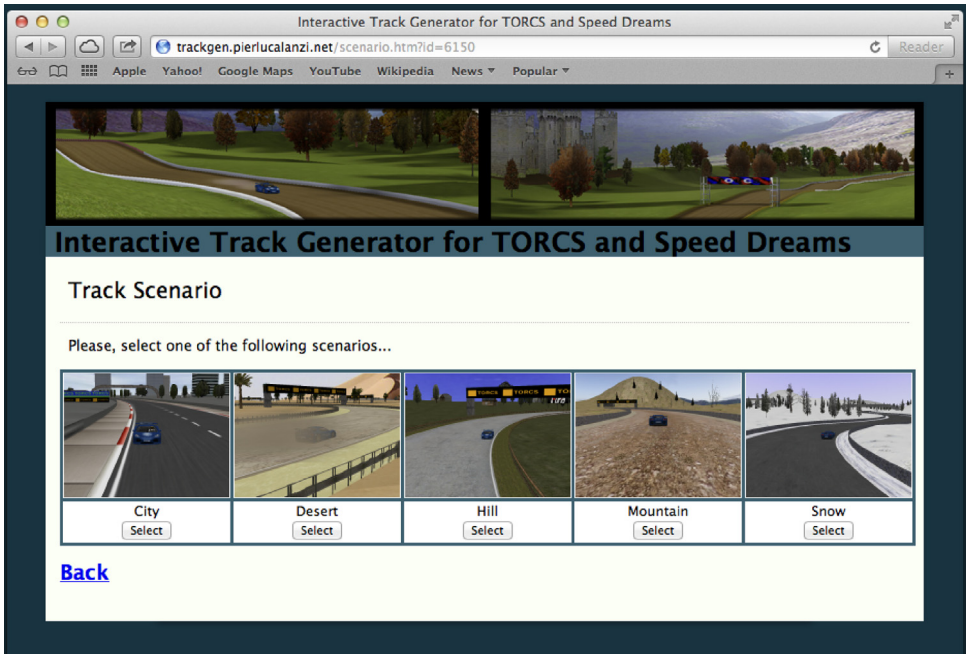

Fig. 4. The download screen where users can select one of the five available scenarios.

number of positive evaluations received (the number of thumbs up) and the number negative evaluations it received (the number of thumbs down); next, two pairs of tracks are randomly selected from the population using tournament selection of size two (lines 9 and 10); Then, crossover (line 11) and mutation (lines 12 and 13) are applied to generate two offspring tracks (o_1 and o_2) from the parents (p_1 and p_2). Next, TORCS/Speed Dreams specific procedures are run (line 15) (i) to check whether the new tracks are feasible (see Section 4), (ii) to compute several track statistics, and finally (iii) to generate the thumbnails needed by the web interface. In this phase, infeasible tracks are discarded while the feasible ones are inserted in the population (line 18) replacing the worst tracks (line 17) so as to update the remote database accordingly. If both offsprings are infeasible, the process is repeated up to ten times; if the system was not able to generate at least one feasible track after ten trials (which rarely happened so far), the population is not modified (line 16); Since TORCS executables tend to be CPU and disk intensive, the evolutionary process schedules the number of active TORCS/Speed Dreams invocations based on the number of cores available and on the number of download requests (which also require separate TORCS/Speed Dreams invocations) to avoid overloading the server. The implementation of the evolutionary algorithm is based on Kumara Sastry's C++ toolbox available from [37] and on the track representation we introduced in [7] (see Section 4)

Algorithm 1. The genetic algorithm – initialization

```
1:  procedure InitGeneticAlgorithm
2:    ClearTheRemoteDatabase();   eliminate existing population
3:    InitTables();   generate initial random population
4:    InitStats();   init all server statistics
5:  end procedure
```

Algorithm 2. The genetic algorithm – steady state evolution

```
1:  procedure SteadyStateGA
2:    while true do   runs forever as a server
3:      if CanSteadyStateEvolve() then
4:                  if enough new users evaluations
5:              runs one cycle of steady state GA
6:        P ← RetrievePopulationFromDB()
7:        trials ← 0
8:        repeat
9:          p1 ← TournamentSelection(P)
10:         p2 ← TournamentSelection(P)
11:         o1, o2 ← Crossover(p1, p2)
12:         Mutation(o1)
13:         Mutation(o2)
14:         trials ← trials + 1
15:       until (Feasible(o1) and Feasible(o2)) or trials>10
16:       if trials≤10 then
17:         DeleteTwoIndividualFromDB()
18:         AddOffspringToDB(o1,o2)
19:       end if
20:     end if
21:   end while
22: end procedure
```

5.3. The scenery generator

The scenery generator (Algorithm 3) is also implemented as a listener process (line 2), hosted on the backend server, that checks the queue of download requests (line 3); when a user requests the download for a track with a selected scenario (Fig. 4), the scenery generator builds the actual TORCS/Speed Dreams package (lines 4–8), then, it uploads the package to the web frontend where the user can download it (line 9). The scenery generator is organized in a pipeline of three main procedures: (i) the elevation heuristic (line 5), (ii) the terrain generator (line 7), and (iii) the track decoration (line 8).

Algorithm 3. The scenery generator

```
1:  procedure SceneryGenerator
2:    while true do   runs forever as a server
3:      if not isRequestQueueEmpty() then   if there is a download
        request
4:        t,s ← GetNextRequestedTrack()   t is the track genome   s is the
        scenery type
5:        et ← ElevationHeuristic(t,s)   generate the plain track
6:                only with elevation information
7:        tt ← AddTerrain(et,s)   add the terrain
8:        ct ← AddGameElements(tt,s)   add the terrain
9:        UploadContentToServer(ct)
10:     end if
11:   end while
12: end procedure
```

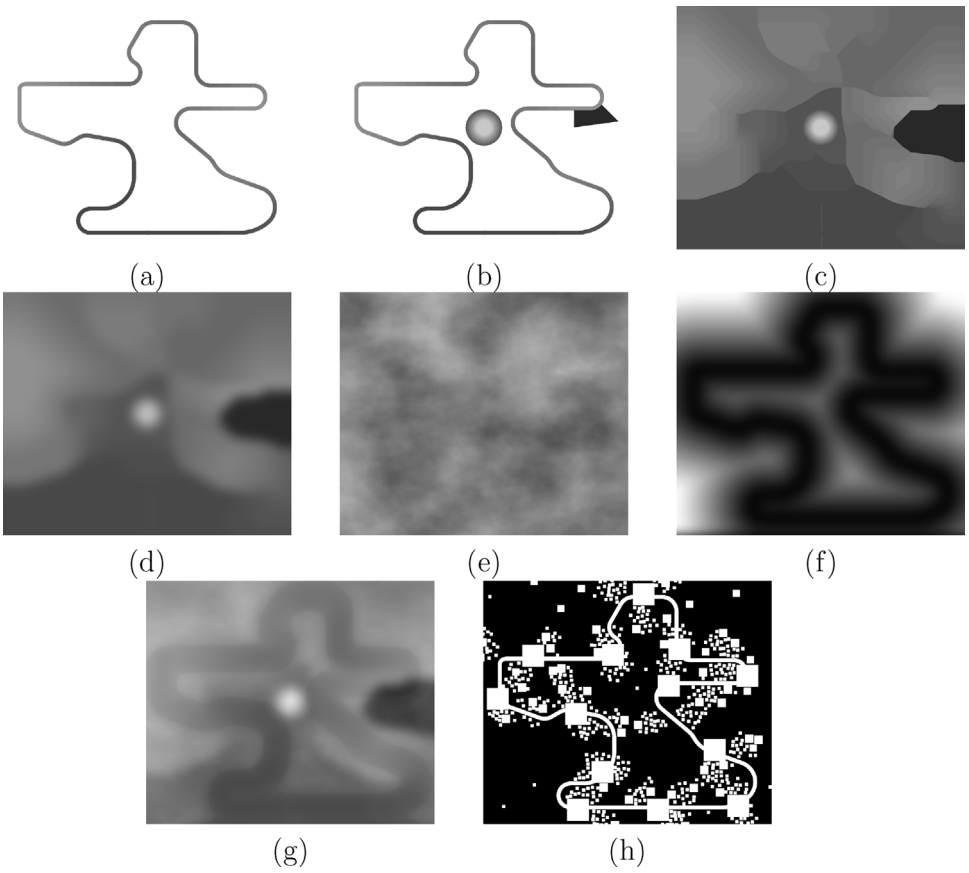

Fig. 5. Scenery generation: (a) track shape with elevation information; (b) enriched with dramatic elements; (c) expanded elevation information to fill the whole map; (d) blurred elevation information; (e) randomly generated Perlin noise; (f) distance map; (g) final elevation map; (h) occupancy map. In figures (a) to (f), light gray identify higher areas of the map while dark gray represent lower areas; in figure (h), white areas are occupied by game elements.

The elevation heuristic takes as input a user request consisting of a track name, its genotype and the name of one of the five available sceneries (city, desert, hill, mountain, snow). At first, it uses the genotype (a vector of real values) to generate a basic TORCS/Speed Dreams track with no assets and no elevation information, represented as a sequence of track segments using the standard XML format. Next, it scans the track and adds elevation information to each segment based on an elevation range which depends on the target scenery: for the city and the desert sceneries, the range is rather narrow so that players will not experience much change in elevation when racing; for the hill scenery, a mid elevation range allows the generation of a track with some nice and smooth elevation transitions; for the mountain and snow sceneries, the elevation range is quite large so that players can experience a wide variety of elevation change while driving. The resulting XML representation of the track (enriched with elevation information) is given to a tool we developed from existing TORCS/Speed Dreams code which generates a 2D gray-scale shape of the requested track (Fig. 5a); lighter colors identify track sections with higher elevations; darker colors identify sections with lower elevation.

The terrain generator starts from the gray-scale shape of the track produced by the previous process (Fig. 5a) and generates an elevation map of the terrain surrounding the track. At first, it applies a heuristic to add *dramatic* elements to the track (e.g., peaks, mountains, gorges, etc.). For instance, it may add an area of relatively high elevation in the center of the map (Fig. 5b) so that nearby sections of the track are not visible or, in mountain/snow scenarios, it may add a gorge on the side of a narrow bend (Fig. 5b). Then, it applies a series of graphic filters and procedures to compute the final elevation map based on the elevation of the track segments and the added dramatic elements. The terrain generator initially expands the elevation information available in the track shape and in the dramatic elements (e.g., the gorges, the mountain) to fill the empty (white) areas of the 2D image received as input (Fig. 5c) and blurs the resulting image to generate smoother transitions (Fig. 5d), i.e., thus smoother meshes. Next, it generates some Perlin noise [38] to increase the scenario diversity (Fig. 5e). Perlin noise [38] is a computer-generated visual effect that is used to represent clouds, fog and other visual effects. In this case, it is used to introduce small random modifications to the previously generated height map. As a results, the smooth surfaces generated during the previous steps will show some small random irregularities that will make the final terrain more realistic. Then, it creates a distance map (Fig. 5f) in which each pixel color represents a weight proportional to its distance from the track (the darker the pixel, the nearer to the map). Finally, the tool uses the weights in the distance map to merge the current map (Fig. 5d) with the Perlin noise (Fig. 5e). The final results is an elevation map with all the information needed to generate the terrain mesh (Fig. 5g).

The positioning of game elements. At the end, the tool adds several types of game elements to decorate the track (e.g., vegetation, buildings, signs, etc.) based on the selected scenery (Section 5.1). This phase outputs a list of assets to be inserted in the track, their position, and the occupancy map (Fig. 5h) which identifies the occupied areas in the final track and it is needed by TORCS/Speed Dreams tools to generate the final package. Fig. 5h shows an example of occupancy map where all the white rectangles identify areas occupied by game elements, e.g., trees (the small white dots), buildings (the larger white rectangles), and signs (crossing the track). The tool applies specific placement heuristics

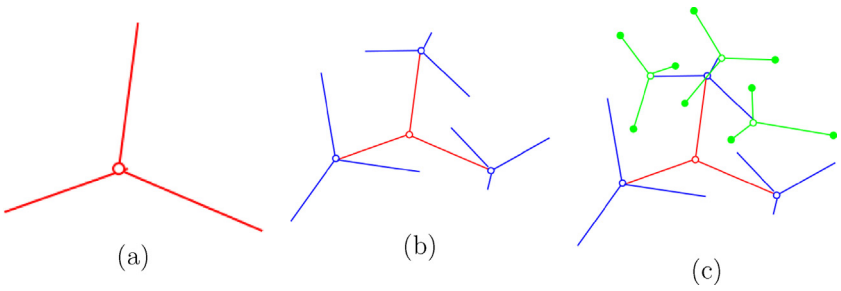

Fig. 6. Placement of vegetation elements.

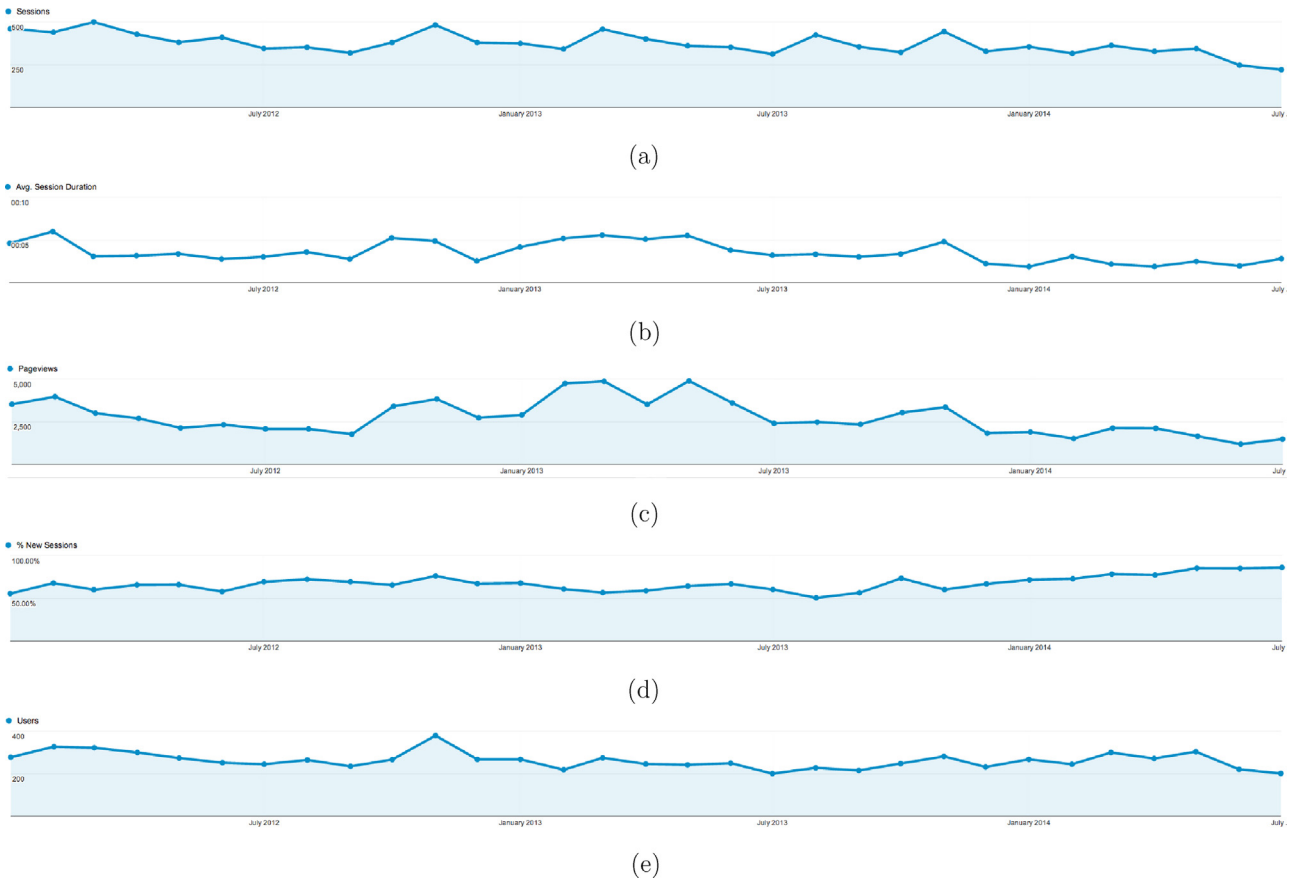

Fig. 7. TrackGen statistics from Google analytics (January 2012–July 2014) plotted per month: number of visits (a), average visit duration (b), pageviews (c), percentage of new users (d), number of users visiting (e).

for each type game element. For instance, the procedure that positions vegetation elements starts from a central point and identifies three directions and three end points (Fig. 6); then, it repeats the same heuristic so as to generate a forest of vegetation elements using the end points. signs are uniformly distributed orthogonally along the track; buildings in city scenarios are positioned using an algorithm similar to the one used for vegetation while in hill scenarios few buildings are randomly positioned to be visible from the track. All the heuristics perform specific checks to avoid nonuniform distributions of game elements (e.g., with all the building in the same area) or overcrowded areas (e.g., area too many buildings or too many trees, track sections with too many signs).

Track generation and upload. When all the information and the data needed to generate the final track are ready, scenery-specific textures are added and TORCS/Speed Dream specific tools are run to generate the actual track that is zipped and uploaded to the web frontend. Two videos available at the TrackGen website[21] and YouTube show how the scenery generation works and examples of the generated scenarios.

6. Statistics

TrackGen has been online since January 2012 with only few short periods of downtime due to maintenance. We tracked the access to the service both using scripts we developed and Google Analytics.[22] From January 2012 until July 2014, according to our data, TrackGen has evolved around 8853 tracks and had around 12,218 download requests for 3869 unique tracks; thus,

[21] Scenery generator videos: http://www.youtube.com/watch?v=3AzjMtRDnBo; http://www.youtube.com/watch?v=Si_u_43HJnM.
[22] *http://www.google.com/analytics/*.

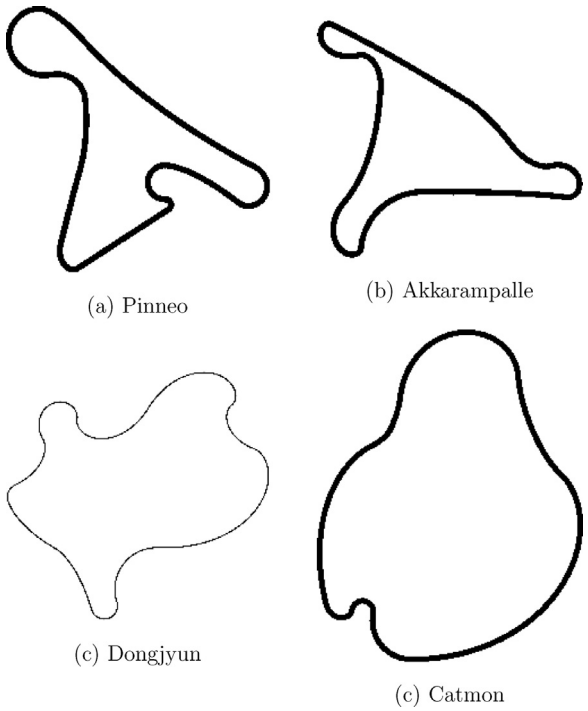

Fig. 8. Tracks with the highest score (a) and (b) and with the lowest score (c) and (d).

TORCS/Speed Dreams packages have been created for 40% of the tracks for one or more scenarios. Note that, after a track is generated its link remains valid forever and we are unable to track the downloads occurring through the direct links. Our statistics show that 1619 different users (tracked using browser cookies) cast their vote for one or more tracks, 4246 distinct users requested the download of a track (i.e., each user downloaded an average of two tracks), and around 6542 votes were submitted. Note that the number of preferences is rather low with respect to the number of generation requests. In fact, the system has no mechanism to enforce voting so it appears that several visitors downloaded tracks but did not leave any vote.

Google Analytics shows that TrackGen was visited around 11,500 times (sessions) from around 7600 unique visitors for a total of around 85,500 page views; each visit lasted for an average of 4 min which is coherent with the time required to generate a track (2–3 min when the request queue is empty). Fig. 7 reports the monthly number of visits (i.e. sessions) (a), the average duration of a visit (b), the number of pageviews (c), the percentage of new users (d), and the number of users including new and returning (f). As can be noticed, there are some peaks. The first one at the beginning when the service was launched and then the activity remains quite stable slowly decreasing until October/December 2012 when there is a peak. The same happens at the beginning of January 2013,[23] in April 2013, in August 2013, and October/December 2013 which interestingly seem related to relevant activities on the official TORCS page: the publication of the new TORCS distribution, v1.3.4 (in October 2012), the preliminary announcement of the new TORCS Endurance World Championship[24] (in December 2012), the showcase of the previous edition of the competition (in January 2013), and the announcement of the "TORCS Endurance World Championship 2013" in (April 2013).

Google Analytics shows that the page visited the most is the download page (accounting for the 25% of the overall traffic corresponding to 24,900 visits), followed by the home page (22% or 19,550 visits); the pages of the most voted tracks and the latest evolved tracks collect 4% of the total pageviews each. The data from Google Analytics suggest that TrackGen users like to browse: in fact, around 10% of the visitors of the homepage browse up to six pages of evolved tracks, thus viewing around 60 tracks.

Fig. 8 shows the two tracks with the highest score, Pinneo (a) and Akkarampalle (b), and the two tracks with the lowest score Dongjyun (c) and Catmon (d). Pinneo (Fig. 8a) is also the most downloaded track and Akkarampalle is the second most downloaded tracks.

7. Conclusions

We presented TrackGen, an online tool for the creation of tracks for TORCS and Speed Dreams, two very popular open-source 3D car racing games. TrackGen combines interactive evolution with procedural content generation to evolve tracks based on users' preferences. Visitors can explore the available tracks, score them, and download tracks rendered using one of the five available scenarios (city, desert, hill, mountain and snow). To our knowledge, TrackGen is the first online collaborative tool for a game community to create content for their game using procedural content generation and interactive evolution. TrackGen has been online since January 2012 and by July 2014 it had 7600 unique visitors, received more than 12,218 download requests, and 85,500 pageviews in total. The analysis of the TrackGen traffic shows peaks corresponding to relevant activity in the TORCS community, which suggest that the tool is used also outside the research community. This was recently confirmed by a request we received from the maintainers of the TORCS project who asked us permission to include some of the tracks evolved by TrackGen in the next TORCS distribution.

TrackGen cannot collect statistics about gameplay (e.g., how long a track was played by the users who downloaded it); unfortunately, these statistics could be available only if the tool was integrated with the official TORCS and Speed Dreams distribution, which represents a possible future direction for the TrackGen project. The system also does not track the number of users simultaneously connected since our current Internet provider does not provide such information (nor to the log of the DB server which is shared among several users). We estimate, given the information collected Google Analytics[25] and what we measured from the generation side, that no more than a couple of users are connected at once – which is coherent with the overall data. Thus we did not have issues of load testing or else. The system so far has been able to manage peaks with several hundreds downloads per day but its usage is still far below what might require some node balancing policy and thus a migration to a cloud solution.

The framework underlying TrackGen is general and could be extended to any task where users' opinions provide a better feedback than an ad-hoc fitness function. In this respect, we plan to extend the framework to the interactive evolution of crowds behavior (a fundamental aspects of many massive online games) whose quality is better evaluated by humans [39–41].

References

[1] Procedural Content Generation, 2014. http://pcg.wikidot.com/
[2] J. Togelius, G.N. Yannakakis, K.O. Stanley, C. Browne, Search-based procedural content generation, in: C.D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C.K. Goh, J.J.M. Guervós, F. Neri, M. Preuss, J. Togelius, G.N.

[23] The small bump is hardly visible using a monthly statistics but it is clearly visible using a finer resolution.
[24] *http://www.berniw.org/trb/*.
[25] http://www.google.com/analytics/.

Yannakakis (Eds.), EvoApplications (1), vol. 6024 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2010, pp. 141–150.
[3] The Open Racing Car Simulator Website, 2014. http://torcs.sourceforge.net/
[4] Speed Dreams – A Free Open Motorsport Sim and Open Source Racing Game, 2014. http://www.speed-dreams.org/
[5] J. Togelius, R. De Nardi, S. Lucas, Towards automatic personalised content creation for racing games, in: Proc. IEEE Symposium on Computational Intelligence and Games, CIG 2007, 2007, pp. 252–259, http://dx.doi.org/10.1109/CIG.2007.368106.
[6] J. Togelius, S.M. Lucas, R.D. Nardi, Computational intelligence in racing games, in: N. Baba, L.C. Jain, H. Handa (Eds.), Advanced Intelligent Paradigms in Computer Games, vol. 71 of Studies in Computational Intelligence, Springer, Berlin, Germany, 2007, pp. 39–69.
[7] D. Loiacono, L. Cardamone, P.L. Lanzi, Automatic track generation for high-end racing games using evolutionary computation, IEEE Trans. Comput. Intell. AI Games 3 (2011) 245–259.
[8] E.J. Hastings, R.K. Guha, K.O. Stanley, Automatic content generation in the galactic arms race video game, IEEE Trans. Comput. Intell. AI Games 4 (2009) 245–263.
[9] L. Cardamone, D. Loiacono, P.L. Lanzi, Interactive evolution for the procedural generation of tracks in a high-end racing game, in: N. Krasnogor, P.L. Lanzi (Eds.), GECCO, ACM, 2011, pp. 395–402.
[10] E. Hastings, R. Guha, K. Stanley, NEAT particles: design, representation, and animation of particle system effects, in: Proc. IEEE Symposium on Computational Intelligence and Games, CIG 2007, Essex, UK, 2007, pp. 154–160, http://dx.doi.org/10.1109/CIG.2007.368092.
[11] K.O. Stanley, R. Miikkulainen, Evolving neural network through augmenting topologies, Evol. Comput. 10 (2002) 99–127.
[12] J. Marks, V. Hom, Automatic design of balanced board games, in: J. Schaeffer, M. Mateas (Eds.), AIIDE, The AAAI Press, 2007, pp. 25–30.
[13] J. Togelius, J. Schmidhuber, An experiment in automatic game design, in: P. Hingston, L. Barone (Eds.), CIG, IEEE, Perth, Australia, 2008, pp. 111–118, http://dx.doi.org/10.1109/CIG.2008.5035629.
[14] C. Browne, Evolutionary Game Design, Springer Verlag, Berlin, Germany, 2011, http://dx.doi.org/10.1007/978-1-4471-2179-4.
[15] P. Avery, J. Togelius, E. Alistar, R.P. van Leeuwen, Computational intelligence and tower defence games, in: IEEE Congress on Evolutionary Computation, IEEE, 2011, pp. 1084–1091, http://dx.doi.org/10.1109/CEC.2011.5949738 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5936494
[16] L. Cardamone, G.N. Yannakakis, J. Togelius, P.L. Lanzi, Evolving interesting maps for a first person shooter, in: C.D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J.J.M. Guervós, F. Neri, M. Preuss, H. Richter, J. Togelius, G.N. Yannakakis (Eds.), EvoApplications (1), vol. 6624 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2011, pp. 63–72, http://dx.doi.org/10.1007/978-3-642-20525-5_7.
[17] M. Frade, F.F. de Vega, C. Cotta, Modelling video games' landscapes by means of genetic terrain programming – a new approach for improving users' experience, in: M. Giacobini, A. Brabazon, S. Cagnoni, G.D. Caro, R. Drechsler, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink, J. McCormack, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, S. Uyar, S. Yang (Eds.), EvoWorkshops, vol. 4974 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2008, pp. 485–490.
[18] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, G.N. Yannakakis, Multiobjective exploration of the starcraft map space, in: G.N. Yannakakis, J. Togelius (Eds.), Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Copenhagen, Denmark, 18–21 August 2010, IEEE, 2010, pp. 265–272.
[19] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, G.N. Yannakakis, C. Grappiolo, Controllable procedural map generation via multiobjective evolution, Genet. Program. Evol. Mach. 14 (2013) 245–277.
[20] W.L. Raffe, F. Zambetta, X. Li, A survey of procedural terrain generation techniques using evolutionary algorithms, in: IEEE Congress on Evolutionary Computation, IEEE, 2012, pp. 1–8, http://dx.doi.org/10.1109/CEC.2012.6256610 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6241678
[21] W.L. Raffe, F. Zambetta, X. Li, Evolving patch-based terrains for use in video games, in: N. Krasnogor, P.L. Lanzi (Eds.), GECCO, ACM, Dublin, Ireland, 2011, pp. 363–370, http://dx.doi.org/10.1145/2001576.2001627.
[22] N. Sorenson, P. Pasquier, Towards a generic framework for automated video game level creation, in: C.D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C.K. Goh, J.J.M. Guervós, F. Neri, M. Preuss, J. Togelius, G.N. Yannakakis (Eds.), EvoApplications (1), vol. 6024 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2010, pp. 131–140.
[23] N. Shaker, M. Nicolau, G.N. Yannakakis, J. Togelius, M. O'Neill, Evolving levels for Super Mario Bros using grammatical evolution, in: IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, 11–14 September 2012, IEEE, 2012, pp. 304–311, http://dx.doi.org/10.1109/CIG.2012.6374170 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6361518
[24] M.G. Friberger, J. Togelius, Generating interesting monopoly boards from open data, in: IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, 11–14 September 2012, IEEE, 2012, pp. 288–295, http://dx.doi.org/10.1109/CIG.2012.6374168 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6361518
[25] A. Jordan, D. Scheftelowitsch, J. Lahni, J. Hartwecker, M. Kuchem, M. Walter-Huber, N. Vortmeier, T. Delbrügger, Ü. Güler, I. Vatolkin, M. Preuss, BeatTheBeat music-based procedural content generation in a mobile game, in: CIG, IEEE, 2012, pp. 320–327 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6361518
[26] D. Plans, D. Morelli, Experience-driven procedural music generation for games, IEEE Trans. Comput. Intell. AI Games 4 (2012) 192–198.
[27] N. Monmarche, G. Nocent, M. Slimane, G. Venturini, P. Santini, Imagine: a tool for generating html style sheets with an interactive genetic algorithm based on genes frequencies, in: IEEE International Conference on Systems, Man, and Cybernetics. IEEE SMC 1999 Conference Proceedings, vol. 3, 1999, pp. 640–645, http://dx.doi.org/10.1109/ICSMC.1999.823287.
[28] H.-S. Kim, S.-B. Cho, Application of interactive genetic algorithm to fashion design, Eng. Appl. Artif. Intell. 13 (2000) 635–644.
[29] C.J. Solomon, S.J. Gibson, J.J. Mist, Interactive evolutionary generation of facial composites for locating suspects in criminal investigations, Appl. Soft Comput. 13 (2013) 3298–3306.
[30] X. Sun, D. Gong, W. Zhang, Interactive genetic algorithms with large population and semi-supervised learning, Appl. Soft Comput. 12 (2012) 3004–3013.
[31] Y. Sato, Voice quality conversion using interactive evolution of prosodic control, Appl. Soft Comput. 5 (2005) 181–192.
[32] A. Brintrup, J. Ramsden, H. Takagi, A. Tiwari, Ergonomic chair design by fusing qualitative and quantitative criteria using interactive genetic algorithms, IEEE Trans. Evol. Comput. 12 (2008) 343–354.
[33] P. Walsh, P. Gade, Terrain generation using an interactive genetic algorithm, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–7.
[34] J. Secretan, N. Beato, Picbreeder: evolving pictures collaboratively online, in: M. Czerwinski, A.M. Lund, D.S. Tan (Eds.), Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, Florence, Italy, 5–10 April 2008, ACM, 2008, pp. 1759–1768, doi: 10.1145/1357054.1357328.
[35] B. Xu, S. Wang, X. Li, An emotional harmony generation system, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–7, http://dx.doi.org/10.1109/CEC.2010.5586210.
[36] S. Risi, J. Lehman, D.B. D'Ambrosio, R. Hall, K.O. Stanley, Combining search-based procedural content generation and social gaming in the Petalz video game, in: M. Riedl, G. Sukthankar (Eds.), Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, 8–12 October 2012, The AAAI Press, 2012 http://eplex.cs.ucf.edu/papers/risi_aiide12.pdf
[37] K. Sastry, Single and Multiobjective Genetic Algorithm Toolbox in C++, Technical Report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL Report No. 2007016, 2007.
[38] K. Perlin, An image synthesizer, in: P. Cole, R. Heilman, B.A. Barsky (Eds.), SIGGRAPH, ACM, San Francisco, CA, USA, 1985, pp. 287–296.
[39] Y.P. Chen, Y.Y. Lin, Controlling the movement of crowds in computer graphics by using the mechanism of particle swarm optimization, Appl. Soft Comput. 9 (2009) 1170–1176.
[40] G. Vigueras, M. Lozano, J. Ordu??a, F. Grimaldo, A comparative study of partitioning methods for crowd simulations, Appl. Soft Comput. 10 (2010) 225–235.
[41] M. Thida, H.-L. Eng, D.N. Monekosso, P. Remagnino, A particle swarm optimisation algorithm with interactive swarms for tracking multiple targets, Appl. Soft Comput. 13 (2013) 3106–3117.