EvoSpace-Interactive: A Framework to Develop Distributed Collaborative-Interactive Evolutionary Algorithms for Artistic Design

Mario García-Valdez[1], Leonardo Trujillo[1], Francisco Fernández de Vega[2], Juan J. Merelo Guervós[3], and Gustavo Olague[4]

[1] Instituto Tecnológico de Tijuana, Tijuana BC, Mexico
[2] Grupo de Evolución Artificial, Universidad de Extremadura, Mérida, Spain
[3] Universidad de Granada, Granada, Spain
[4] Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada BC, Mexico
mario@tectijuana.edu.mx, leonardo.trujillo@tectijuana.edu.mx
fcofdez@unex.es, jjmerelo@geneura.ugr.es, olague@cicese.mx

Abstract. Currently, a large number of computing systems and user applications are focused on distributed and collaborative models for heterogeneous devices, exploiting cloud-based approaches and social networking. However, such systems have not been fully exploited by the evolutionary computation community. This work is an attempt to bridge this gap, and integrate interactive evolutionary computation with a distributed cloud-based approach that integrates with social networking for collaborative design of artistic artifacts. Such an approach to evolutionary art could fully leverage the concept of memes as an idea that spreads from person to person, within a computational system. In particular, this work presents EvoSpace-Interactive, an open source framework for the development of collaborative-interactive evolutionary algorithms, a computational tool that facilitates the development of interactive algorithms for artistic design. A proof of concept application is developed on EvoSpace-Interactive called *Shapes* that incorporates the popular social network Facebook for the collaborative evolution of artistic images generated using the Processing programming language. Initial results are encouraging, *Shapes* illustrates that it is possible to use EvoSpace-Interactive to effectively develop and deploy a collaborative system.

Keywords: Interactive Evolution, Collaborative Design, Evolutionary Art, Cloud Computing

1 Introduction

Evolutionary Algorithms (EAs) where conceived as general techniques capable of addressing a large set of hard optimization problems [6]. Yet, their possibilities as a source of creativity were soon devised and applied to art, design

and music, to name but a few [15]. Different researchers noticed both, their inherent randomized search process conducting to different solutions for a single problem when repeatedly running the algorithm, but also their feasibility when configured to generate novel designs from an aesthetic point of view. Thus, EAs were soon employed for generating design and art concepts [1].

Nonetheless, a number of problems has been noticed when dealing with creativity and Evolutionary Algorithms [9]. Among them, we can find the difficulties when deciding how to encode design or art concepts within a data structure -which give rise to individuals and populations- as well as the definition of useful genetic (search) operators over the chromosome. A particular hard problem is how to evaluate the quality of individuals within the population when aesthetic concepts are considered. The problem is that creativity and aesthetic principles are not clearly understood, which keeps from accurately defining a function that can properly measure it. Therefore, researchers soon resort to human brains to perform that specific task, thus defining Interactive Evolutionary Algorithms (IEAs): standard EAs whose fitness evaluations are performed by human beings in an interactive fashion. Thus, the main loop of the EA includes the human intervention for quality assessment of evolved solutions. Nevertheless, an inherent drawback arises from the very nature of the model, namely, human fatigue caused by repeated interaction -and some authors have already tried to address this [5]. In any case, IEAs have demonstrated their capability for effectively producing art and design [1, 14, 16]. We should also mention that typically artists with a strong background in computer science, or computer scientists with art/design interest, are the researchers involved in this area, since it can be difficult for an artist without programming capabilities to enter this research domain.

Therefore, a disadvantage remains when addressing art and design by means of EAs, when compared with standard optimization problems. Researchers have developed over the years not only improved versions of the basic operators, models or meta-models for EAs, they have also embedded these improvements and their algorithms within toolboxes that allow other researchers to easily launch an EA when solving an optimization problem. This is exactly the opposite for art and design: no EA tool specifically aimed to this field has been design yet. Researchers have typically implemented specific versions fitted to the problem they try to address, but no effort is aimed at developing a standard EA-based tool for Art and Design.

Some desirable properties for a specific IEA toolbox useful for artists are:

1. An easy process for deploying individuals capable of generative art/design. Given that standard users of this toolboxes will be artists with no background in computer science, an easy means for encoding art concepts should be provided,without the need to understand how to encode an EA;
2. The direct influences among different artists should not only be allowed, but encouraged, in a collaborative process.

The art world grows on a network of influences that allows ideas to travel between minds. Authentic memetic evolution of artistic ideas arises from an

artist's mind and hands, and this should be automatically promoted by a specific toolbox: memetic evolution without any explicit need of algorithm encoding. Thus, the term memetic evolution, as originally defined by Richard Dawkins [3], relying on both minds and computers would be finally a main component of the evolutionary process.

This work provides the first tool-set that explicitly addresses these issues, a collaborative and interactive EA framework that will easily allow artists to evolve their creations while interacting with a network of artists and friends working together around the world. The remainder of this paper proceeds as follows. Section 2 presents related work on the topic of Interactive Evolution. Then, Section 3 presents the computational platform on which the main proposal of this work is developed. The proposed collaborative interactive EA is presented in Section 4. The experimental work and results are presented in Section 5. Finally, a concluding remarks are provided in Section 6.

2 Related Work

Interactive evolution (IE) incorporates human preferences into the selective pressure that guides the search [15, 13]. Specifically, IE poses an open-ended evolutionary search, where the objective function of traditional EAs is replaced by a subjective evaluation carried out by a human user of the system. In one sense, this open-ended nature of IE broadens the range of feasible solutions that can be reached by the search, promoting diversity and exploration of design space. In fact, some of the earliest EAs were open-ended interactive systems, such as the well-known Biomorphs program [4]. For instance, IE has been combined with the recently proposed Novelty Search algorithm [17], to promote and exploit these properties.

IE is an active area of research, and the approach has been used in a wide variety of applications and problem domains. However, in this work, emphasis is given to IEAs that were designed to evolve artistic artifacts. In particular, collaborative systems where many users interact and evaluate an evolving population, thus guiding the search based on an aggregate of subjective preferences and considerations. Such systems can be referred to as Collaborative IEAs or C-IEAs. What follows is not intended as a comprehensive survey, only a review of the most relevant contributions to the present work.

An early example of a web based interactive system is the work by Langdon [8], which evolves fractal representations of virtual creatures. It proposed a distributed EA using a global population that resides on a central web server and distributes portions of the population to remote clients using Javascript. Users evaluate individuals locally and those a user prefers are returned to the server and can be distributed over the web. Similarly, Secretan et al. [12] and Clune and Lipson [2] use web-based IEAs to evolve artistic artifacts using a generative encoding, compositional pattern producing networks. Images are evolved in [12] and 3-D sculptures in [2] . In both cases, user (connected clients) collaboration is encouraged. Both works offer webpages (see Picbreeder.org

and EndlessForms.com), where users can select or create random individuals and evolve lineages of artifacts based on their preferences. In these systems a user can take a previously evolved artifact and continue the search process himself, building upon a previous evolutionary design in a sequential manner. Therefore, evolved artifacts can be the product of a collaborative search process. Another feature is that evolved artifacts can be rated by users, and since users can create individual accounts, the ratings provide a way to rank users, or to select previously evolved artifacts based on the particular style of each user. Furthermore, the collaborative process is captured by the system, since it is possible to visualize how, and when, different users influenced a genetic lineage. Kowaliw et al. [7] present another recent example, evolving ecosystemic models, a generative encoding based on multi-agent systems, that generate high quality artistic drawings. Users visit a website (`http://www.csse.monash.edu.au/~cema/evoeco/`) and interact with a Java applet, after which they can choose to add evolved images to a central collection, such that other users can see the resulting images, or use the images as seeds for their own IE design.

The present work builds on previous proposals and extends the C-IEA approach. First, it promotes collaborative evaluations of artistic artifacts in a dynamic and parallel manner, instead of the sequential approach followed in [12, 2]. Second, it incorporates explicit user interactions by encouraging the use of social networking. Third, it facilitates the ability to save and share promising artifacts. Finally, it emphasizes the use of a cloud-based model, with support for multiple computing devices.

3 EvoSpace

EvoSpace is a population store for the development of evolutionary algorithms that are intended to run on a cloud computing model. It is designed to be versatile, since the population is decoupled from any particular evolutionary algorithm. Evospace is asynchronous, client processes, called EvoWorkers, dynamically and asynchronously interact with the EvoSpace store and perform the basic routines of an evolutionary search. EvoWorkers can reside on remote clients or on the platform server itself.

EvoSpace consists of two main components. First, the EvoSpace container that stores the evolving population. The second component consists of the remote clients called EvoWorkers, which execute the actual evolutionary process, while EvoSpace is only a population repository. Figure 1 illustrates the main components and dataflow within EvoSpace.

3.1 The EvoSpace container.

EvoSpace is based on the tuplespace model, an associatively addressed memory space shared by several processes. A tuplespace can be described as a distributed shared memory (DSM) abstraction, organized as a *bag* of tuples. A tu-

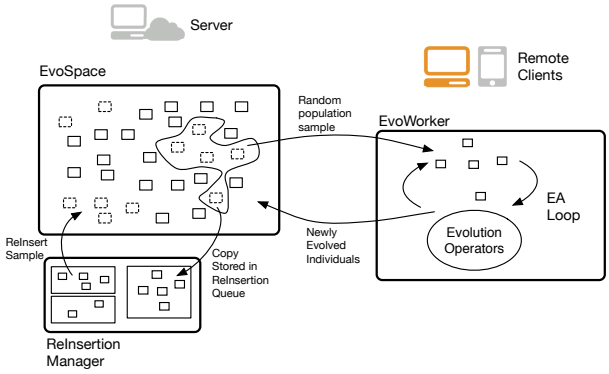

Fig. 1. Main components and dataflow within EvoSpace.

ple t is the basic tuplespace element, composed by one or more fields and corresponding values. In this model, the basic operations that a process can perform is to insert or withdraw tuples from the tuplespace. EvoSpace is composed by a set of objects ES and a set of interface methods provided by a central server. Objects can be withdrawn, processed and replaced into ES using a specified set of methods. However, EvoSpace is different from other tuplespace implementations in the sense that retrieving and reading objects from ES are random operations. Individual objects are not of high interest when accessing ES, neither is retrieving objects based on search criteria. Therefore, EvoSpace offers the following interface methods.

Read(n): This method returns a random set A of objects from ES, with $|A| = n$ and $A \subset ES$, if $n < |ES|$, the method returns ES otherwise.

Take(n): Returns a random set A, following the similar logic used for $Read()$. However, in this case the sequence of $Take()$ operations provide a temporal dimension to the dynamics of set ES. We can define ES_i as the set at the moment of the $i-th$ $Take()$ operation and A_i as the output. The contents of EvoSapce are then given by $ES_{i+1} = ES_i \setminus A_i$; i.e., the objets taken are effectively removed from ES. The objects taken are also copied to a new set S_i of *sampled objects* and stored within a temporary collection $\mathscr{S}$ on the server, implemented as a priority queue. Sets $S_i \in \mathscr{S}$ can then be reinserted to ES if necessary.

ReInsert(i): This method is used to reinsert the subset of elements removed by the $i-th$ $Take()$ operation, such that the contents of EvoSpace are now $ES \cup S_i$ if $S_i \in \mathscr{S}$ and ES is left unchanged otherwise.

Insert(A): This method represents the union operation $ES \cup A$.

Replace(A,i): Similar to $Add()$, however set A should be understood as a replacement for some $S_i \in \mathscr{S}$, hence $|A| = |S_i|$, but the objects in A can be different (evolved) objects from those in S_i. Moreover, if S_i exists it is removed from $\mathscr{S}$. However, if S_i does not exist this means that a $ReInsert(i)$ operation preceded it, this increases the size of ES.

Remove(A): This method removes all of the objects in A that are also in ES, in such a way that the contents of EvoSpace are now set to $ES \cup (A \cap ES)$.

Individuals. Until now, we have assumed that the objects in ES represent individuals in an EA. Explicitly, the objects in ES are stored as *dictionaries*, an abstract data type that represents a collection of unique keys and values, with a one to one association. In this case, keys represent specific properties of each object and the values can be of different types, such as numbers, strings, lists, tuples or other dictionaries.

The EvoSpace Server Processes. On the server side, a process called `EvoSpaceServer` is executed, which creates and activates a new EvoSpace container object and waits for requests to execute interface methods. Additionally, on the server three more processes are executed, these are: `InitPopulation`, `ReInsertionMgr` and `EvolutionMgr`. `InitPopulation` is executed once, its goal is initialize the population by adding *popsize* random objects. The function that creates new individuals depends on the problem and the representation used. `ReInsertionMgr` is used as a failsafe process that periodically checks (every *wt* seconds) if the size of the population in ES falls below a certain threshold *min* or if the time after the last reinsetion is greater than $next_r$. If any of these conditions are satisfied, then *rn* subsets $S_i \in \mathscr{S}$ are reinserted into ES using the *ReInsertOld*() method. Finally, `EvolutionMgr` periodically checks if a termination condition is satisfied, which is checked by the *isOver*() method. This method can be implemented according to the needs of the evolutionary search.

EvoSpace Clients: EvoWorkers. The `EvoWorker` process is straightforward, it requests a set of objects A_i from the ES container. Afterwards, the *Evolve*() function is called where the actual evolutionary cycle is performed. In this scenario, A_i can be seen as a local population on which evolution is carried out for g generations. The result of this evolution is then returned and reinserted into ES, afterwards the EvoWorker can request a new set from ES and repeat the process. Otherwise, each EvoWorker could specialize on a particular part of the evolutionary process, such as selection, evaluation or genetic variation; an approach not taken in the present paper.

Implementation. Individuals are stored in-memory, using the Redis key-value database. Redis was chosen over a SQL-based management system, or other non-SQL alternatives, because it provides a hash based implementation of sets and queues which are natural data structures for the EvoSpace model. For example, selecting a random key from a set has a complexity of O(1). The logic of EvoSpace is implemented as a Python module exposed as a Web Service using cherrypy and Django HTTP frameworks. The EvoSpace web service can interact with any language supporting json-rpc or ajax requests. The EvoSpace modules and workers in JavaScript, JQuery and python are available with a Simplified BSD License from `http://evospace.org`.

4 A Framework for Collaborative Interactive Evolution

The goal of this work is to develop an open source framework for Web and Cloud-based C-IEA systems, using current web standards and libraries for mobile devices. The framework is called EvoSpace-Interactive, its main components are depicted in Figure 2. Developers of C-IEA applications are liberated from the need of designing and programming a platform for distributed user collaboration. Only three components of the framework must to be defined for each application, marked with double lines in Figure 2; these are: an *individual* representation; a *processing script* that renders each individual; and a *worker* script that encodes the evolutionary operators will need to be defined according to the representation and problem domain. However, in future versions of the framework much of this work could be predefined, but also left open for advanced users to change as they require. What the framework offers for free is: a central repository for the population implemented as an EvoSpace service; a Web Application script implemented using Django, a mature full stack Web Framework with a BSD license developed in Python. This application is responsible for user authentication and session handling through the popular Facebook social network using the OAuth 2.0 protocol. Also, the storage of collections, where users can store individuals they like to share with friends, is persisted using the PostgreSQL DBMS. Most of the interactive functionality is programmed in the client side using Javascript libraries. The communication between components is implemented using json-rpc, a lightweight remote procedure call protocol and common ajax and http transactions. Overall functionality is decomposed in specialized services, adding flexibility to the framework since services can be interchanged. The framework is build using only open source components from libraries to servers. Users interact with the system through a GUI implemented on a Responsive Web Design (RWD) front end framework, an approach to web design in which a graphical user interface is crafted to provide a satisfactory viewing experience in a range of mobile devices. This approach will enable designers to tailor the look and feel of the application with minimum intrusion, only changing CSS definitions. Current functionality from the user perspective is described next.

4.1 User Interface

The users interact with the web interface depicted in Figure 3, which is composed of five elements. First, at the top left corner user login and authentication. Users can login with their Facebook account or participate as anonymous users. Second, if a user chooses to login a list of Facebook friends that have also linked their account with the C-IEA application is presented on the left, to encourage users to interact with the system.

The third element is a central *Wall* area, where a population sample of n individuals is shown to the user. These are n random individuals taken from the EvoSpace server. Here, the user can interact with the system in two ways. He can click on the individuals he prefers, a clicked image is highlighted and this

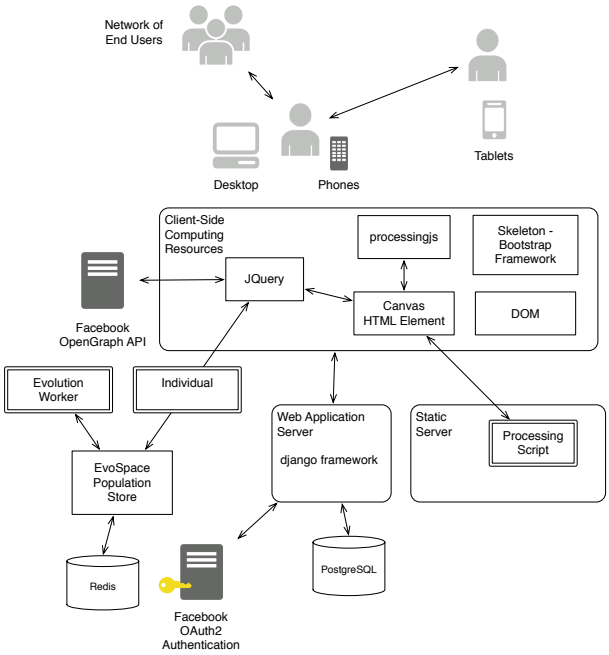

Fig. 2. Conceptual design of the proposed C-IEA.

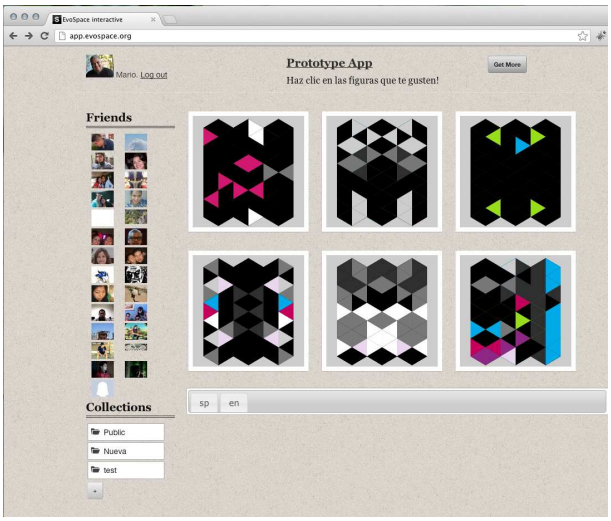

Fig. 3. User interface of the EvoApp C-IEA.

counts as a ”like" for the individual (this is further explained in the following section). Additionally, a user can choose to add an image to one of their *Collec-*

tions. A collection is a special directory to store individuals a user prefers and wishes to save.

After the user finishes interacting with the current crop of individuals on the Wall, he can choose to retrieve a new sample from EvoSpace. This is done with the fourth element of the interface, located at the top of the screen, the *GetMore* button. The button returns the current group of individuals to EvoSpace, and brings back a new one. Moreover, each time a user performs a *GetMore* click, it triggers a server-side *Breeding* event, this event can be used to trigger an application dependent Breeding process.

The fifth element of the interface is shown at the bottom left corner, the *Collections* section. The user can create several collections, to group and organize his favorite artifacts. Moreover, a user can browse the content of each collection and from there share images through the social network. When a user browses over an individual a detail pane shows how many users have liked the individual. The pane also includes a link to the individual's details, the parents, genetic operators that created it, and genealogy information.

4.2 Individual Representation

Individuals are represented internally as a dictionary, as mentioned in Section 3. The basic properties stored for EvoSapce-Interactive applications are: a unique `id`; a user defined `chromosome`; the number of times the individual has been selected in a sample and returned to the population, stored in property called `views`; the genetic operators that generated the individual; ids of the parents; `current Fitness` that stores the most resent fitness value; and a `fitness` dictionary where each key is a concatenation of a `userid`, a `timestamp` and a numerical value that represents the rating given by the corresponding individual. A UML representation of an individual is given in Figure 4.

4.3 Processing Language and HTML Canvas Element

Processing is a programming language and development environment initially created to serve as a software sketchbook and as a tool to teach fundamentals of computer programming within a visual context. Currently is used by artists, designers, architects, and researchers for visualization applications, games and interactive animations projects [11]. Processing is a subset of Java directed to novice programmers and generative artists [10], which are the intended users of the EvoSpace-Interactive framework. As a complement there is a javascript library *processingjs* that allows Processing scripts to be run by any HTML5 compatible browser. Processing scripts are responsible of rendering individuals which can involve animations, sound or even interactive artifacts. Before calling the `draw()` method of the processing script initial place holding or fallback parameters are replaced with an individual's chromosome. Each individual's script has its own Canvas entity, defined by the HTML5 standard as an element that provides scripts with a resolution-dependent bitmap canvas which can be used

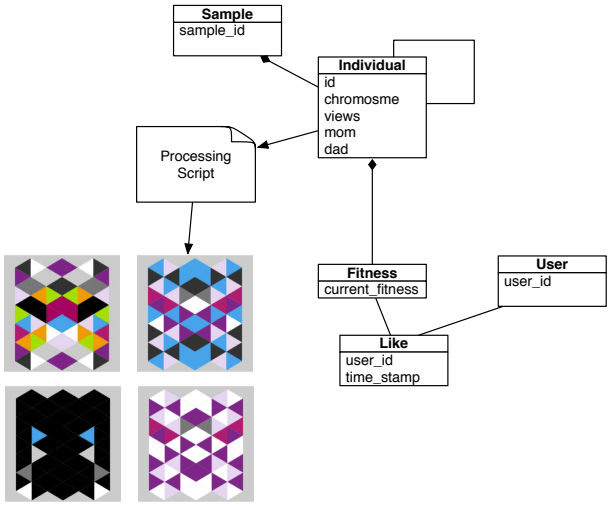

Fig. 4. Individual internal representation

for rendering graphics on the fly. Although the combination of an HTML5 Canvas element and a Processing script is supported by default, other combinations could be used. For instance, images, embedded audio, or other libraries capable of drawing in the Canvas. Also, a fallback implementation must be considered for applications intended for non-HTML5 capable browsers.

5 Experiment: Shapes Application

As a proof of concept a C-EIA application was implemented with the EvoSpace-Interactive framework. The application is called *Shapes*, and implements each EvoSpace-Interactive component as follows.

5.1 Individual Representation and Breeding Process

In *Shapes*, individuals represent a two dimensional 11 by 6 array of equilateral triangles, these arrays are sometimes used in Op-Art style paintings. Each triangle has a color drawn from a twelve color palette. The array is represented by a 66 element chromosome vector $\mathbf{v} = (v_1, ..v_{66})$, with $v_i \in \{1, 2, ..11\}$. The background of the painting is Light Gray, this can give the effect of a missing triangle when it has the same color. A processing script is used to render a static version of the image. The breeding process uses tournament selection of size 6 to select two individuals from EvoSpace, and generates two offspring. The offspring replace the worst individuals from both tournament groups. Crossover operators are used with crossover rate of 1, these are vertical and horizontal one-point crossovers. Several mutations are used with a mutation rate of 0.3, these are: (1)

single *point mutation*; (2) vertical and horizontal *mirrors* at a random point; (3) *shuffle* that gives a new permutation of the chromosome.

5.2 Fitness Evaluation

The fitness of each individual takes into account the evaluation given by several users. In *Shapes*, users can only give positive evaluations explicitly when they select an individual, a *Like*. When a user evaluates a sample of individuals, some (or all) of them will not receive a vote, in each case the `views` property will be incremented by 1. For instance, if an individual has a high number of views with with only two likes, he is worse than an individual with two views and two likes. The ratio *Likes*/*Views* is more informative, but it does not distinguish between an individual with many views and another with only one view if they both have zero likes; also views must be >=1 to avoid dividing by zero. Fitness, therefore, is given by $(Likes + 1)/(Views + 1)$.

5.3 System Setup

The population was initialized with 500 randomly generated individuals. Every second sample returned from users triggers a Breeding process is executed. `ES.ReInsert()` method was called when the sample queue reaches a threshold of 20 samples. A maximum number of concurrent 40 users was expected. The system was installed in a virtual private server with 500 MB of RAM. The http Server used was Gunicorn behind a Nginx http proxy. A simple call for participation was issued by two authors in their personal Facebook and Twitter accounts.

5.4 Results

In nearly two weeks of operation, there is a total of 70 active users, users who gave permissions to the *Shapes* application and haven't removed it from their Facebook account. Facebook dashboard reports that 74 percent of users accepted the permission request to use their credentials to login onto *Shapes*. Participation of anonymous users was permitted, but their number was not recorded. Basic instructions we're shown in the landing page, but part of the functionality was left to be explored by users. An auto-increment id was assigned to each individual, after two weeks the highest id number is 8379. A total of 17449 samples were taken from the EvoSpace server after the two week trial. A sample of artistic artifacts in a collection is depicted in Figure 5.

6 Conclusions

Initial results are encouraging, the EvoSpace-Interactive framework was successfully used to deploy a C-IEA that users accepted and used to design artistic

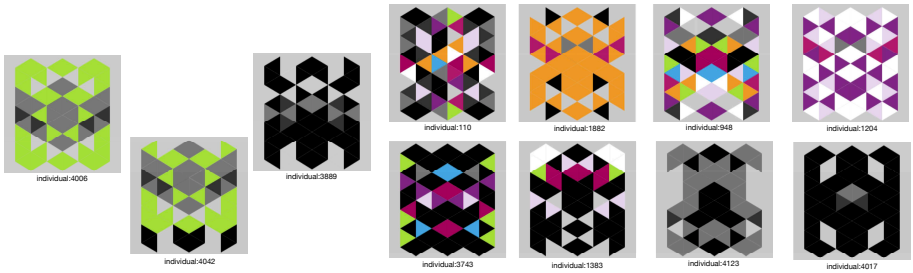

Fig. 5. Artistic artifacts in a collection

artifacts. Since the system integrates with a popular social network, the framework promotes the collaborative evolution of artistic memes, leveraging the insights of multiple users in a parallel and asynchronous manner. The platform enables the collaborative assignment of fitness and provides researchers with relevant contextual information about the process, that can be considered not only in the evaluation step, but also by the system as a whole, allowing a better understanding of user behavior and preferences. Moreover, through the use of the Processing language, and easy to develop representation of artistic artifacts is possible, the possibilities of which were not yet pushed to the limit by the simple Shapes application, the main topic of future work and research. Nonetheless, the paper presents the first attempt to build a tool that facilitates the development and deployment of C-IEA for evolutionary art.

Acknowledgments. This research was supported by DEGEST (Mexico) Research Project "*EvoSpace: Un gestor de poblaciones para computación evolutiva interactiva distribuida*". Additionally, second author supported by CONACYT (Mexico) Basic Science Research Grant No. 178323, "*Predicción de Rendimiento y Dificultad de Problemas en Programación Genética*".

References

1. P. Bentley. An introduction to evolutionary design by computers. In P. J. Bentley, editor, *Evolutionary Design by Computers*, chapter 1, pages 1–73. Morgan Kaufman, San Francisco, USA, 1999.
2. J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pages 144–148, 2011.
3. R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 1976.
4. R. Dawkins. *Climbing Mount Improbable*. W.W. Norton & Company, 1996.
5. M. Frade, F. Fernandez de Vega, and C. Cotta. Evolution of artificial terrains for video games based on accessibility. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekart, A. I. Esparcia-Alcazar, C.-K. Goh, J. J. Merelo, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis, editors, *EvoGAMES*, volume 6024 of *LNCS*, pages 90–99, Istanbul, 7-9 Apr. 2010. Springer.
6. K. A. D. Jong. *Evolutionary computation - a unified approach*. MIT Press, 2006.

7. T. Kowaliw, A. Dorin, and J. McCormack. Promoting creative design in interactive evolutionary computation. *Evolutionary Computation, IEEE Transactions on*, 16(4):523 –536, 2012.
8. W. B. Langdon. Global distributed evolution of l-systems fractals. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming, Proceedings of EuroGP'2004*, volume 3003 of *LNCS*, pages 349–358. Springer-Verlag, 5-7 April 2004.
9. J. McCormack. Open problems in evolutionary music and art. In *Proceedings of the 3rd European conference on Applications of Evolutionary Computing*, EC'05, pages 428–436, Berlin, Heidelberg, 2005. Springer-Verlag.
10. M. Pearson. *Generative Art*. Manning Publications, pap/psc edition, July 2011.
11. C. Reas and B. Fry. *A programming handbook for visual designers and artists*. The MIT Press, 2007.
12. J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evol. Comput.*, 19(3):373–403, Sept. 2011.
13. Y. Semet. Evolutionary computation: a survey of existing theory. Technical report, University of Illinois, 2002.
14. K. Sims. Artificial evolution for computer graphics. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 319–328, New York, NY, USA, 1991. ACM.
15. H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of IEEE*, 89(9):1275–1296, 2001.
16. S. Todd and W. Latham. *Evolutionary art and computers*. Academic Press, 1992.
17. B. G. Woolley and K. O. Stanley. Exploring promising stepping stones by combining novelty search with interactive evolution. *CoRR*, abs/1207.6682, 2012.