Xovos: Web Design through Evolution

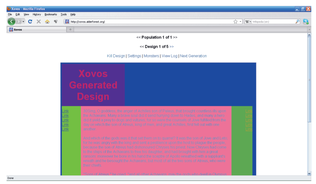
Eric Marcarelli
Quinnipiac University

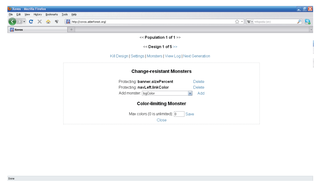
Introduction

Xovos is a program that develops web designs in an evolutionary environment with user interactions and preferences as the selection criteria. The project consists of five basic parts. The user interface allows the user to interact with the program. A set of genes makes up a chromosome that describes a design. An offspring generator mates the designs from the current generation to form the next generation. The design generator parses the design chromosomes to create the visual representations of the design. Finally, environmental preferences (also called monsters) allow users to set basic criteria for designs that will be automatically selected against. In my research I have not been able to find anyone else who has tried to apply evolutionary programming to web design in this way. The closest related work is on evolving programs that are designed for specific tasks and are not visually oriented.

User Interface

The interface to Xovos is a web page that allows the user to browse through populations and designs, make design selections, customize settings, add monsters, view the log, and generate the next generation.





Data Representation

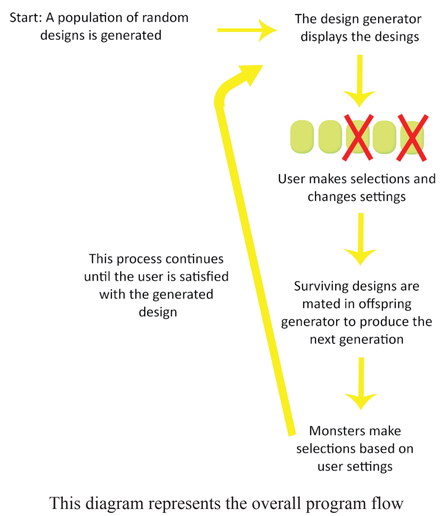
Users can create any number of populations, and each population can contain an arbitrary number of designs. Designs consist mostly of their genetic chromosome. Chromosomes are built out of genes, which store an identifier, value, and pointer to a function that can randomize that value. The work of Ivory, Sinham, and Hearst (2001), which proposed a set of empirical metrics for web design, was helpful in deciding the particular genes I included. More important than the particular gene choices was the fact that the system could easily be expanded to include any number of new genes.

My choice of population structure was inspired by Chu & Dudley's (1993) work. In their paper they consider the effects of population size and structure on the results that are produced by a genetic algorithm. They determine that groups with small subpopulations are better able to solve problems where there may be multiple good solutions, or "peaks." The single large group could find one peak and gravitate around it, oblivious to a larger peak that might exist, and the subpopulations are less likely to all find the same peak. This structure also makes the system better able to survive any damage caused by recombination or mutation. One drawback to the current implementation of populations is the inability to recombine them. While they can be used to "save" good results before making a risky selection, the isolated population is permanently shielded from future improvements as well as damage.

Design Generator

The design generator reads design chromosomes and generates the HTML and CSS for the visual design. One of the most challenging aspects of the project was deciding how to generate designs that would be viable HTML from a set of genes. The approach I took has some limitations, but it always produces a valid web page. The generator is built around a default design. Based on the genetic makeup of the chromosome, certain parts of the page are displayed or not, and everything that is displayed can have its colors, fonts, and sizes modified. While the first pass at this method leaves the pages with a similar layout, there is no reason, aside from time constraints, that it could not be expanded to more flexible structuring.

One interesting aspect of the design generator is that certain genes turn on and off entire sections of the page, so that dormant qualities in a chromosome could suddenly become expressed through one gene swap or mutation. This was inspired by the work of Yu (2007) who investigated how programs can develop the ability to successfully adapt to changes by first evolving them to produce correct answers according to one fitness function, then switching to a different function.



This diagram represents the overall program flow

Offspring Generator

The offspring generator is the part of the project that drives the evolution. The generator takes the current populations of designs and creates a new set of populations that become the next generation. The algorithm for generating the next generation is described below.

numPerGeneration = the number of designs to produce per population.
splitThreshold = the number at which a population will be split to start a second population.
numSwaps = the number of genes that a design will swap per mate.
numMates = the number of designs each design will swap genes with.
mutationChance = the chance that a mutation will occur in one of the genes of a chromosome.
i = number of current offspring
n = population size

Do the following for each population while numPerGeneration is less than i:

> Copy the (i mod n)th member of the population into the new population.
>
> Let j be the number of gene swaps completed. While j < numSwaps, repeat:
>
> Randomly select a member of the old population
>
> Randomly select a gene
>
> Check that there is not a monster protecting that gene from change.
>
> If not, replace the gene in the copied chromosome with the gene from the selected chromosome
>
> Calculate a number from 1 to 100. If that is number is less than mutationChance, randomly select a gene and randomize its value.

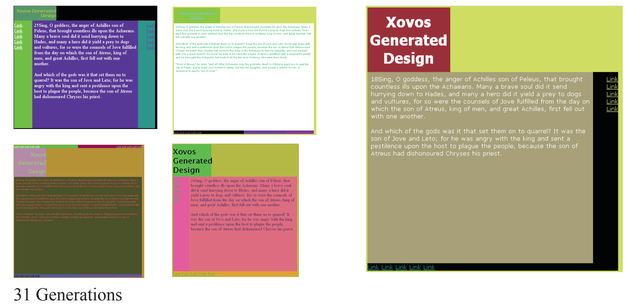
Because the diversity of a population will decrease over time, the biggest changes will occur in the first 10-20 generations, depending on population size. After that the final structure of the design should start to plateau and improve gradually through this low level of mutations.

Results

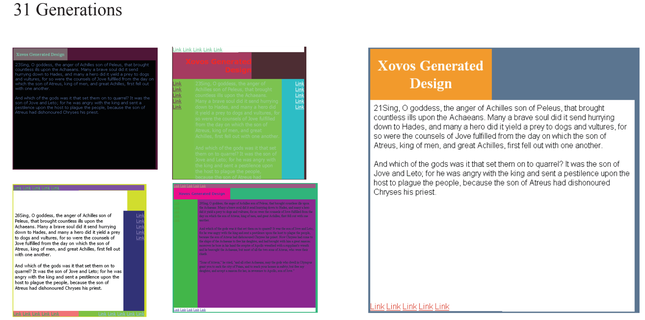
Xovos is capable of producing improvements in designs after about 20-30 generations given a reasonable series of user selections. This can be most easily demonstrated by comparing the final designs produced by Xovos to their random starting points.

While judgments of designs are inherently subjective, it seems obvious that the end results show improvement from the random beginnings. The color schemes are more unified, balanced, and aesthetically pleasing. The amount of chaos in the designs is also reduced and they portray a more polished feel than the starting designs.

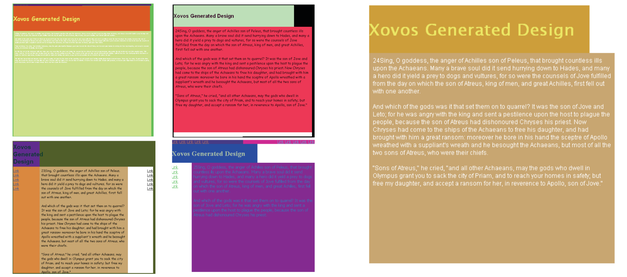
Three example designs produced by Xovos are shown below. On the left side are the random starting designs, and on the right are the final designs. The number of generations is indicated below each set.



31 Generations



28 Generations



78 Generations

References

Chu, P. H. and Dudley, S. A. 1993. "The effect of population structure on the rate of convergence of genetic algorithms." In Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice.

Ivory, M. Y., Sinha, R. R., and Hearst, M. A. 2001. "Empirically validated web page design metrics." In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.

Langdon, W. B. 1995. "Evolving Data Structures with Genetic Programming." In Proceedings of the 6th international Conference on Genetic Algorithms.

Yu, T. 2007. "Program evolvability under environmental variations and neutrality." In Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation .