David Richards
CS4800 Algorithms and Data
Fall 2011
Professor Lieberherr
December 14th, 2011

Algorithmic Invention and Algorithmic Claim Evaluation with SCG

Overview

The Scientific Community Game (SCG) provides a powerful platform for learning about and implementing algorithms. The platform allows algorithmic implementations of the solution to a problem, known as avatars, to face off against various other avatars in an even playing field. This form of competition encourages an evolutionary improvement of the avatars as implementers view the strengths and weaknesses of both their avatar and the avatars they play against.

While SCG is primarily a software driven platform, the basic concepts can also be abstracted and played manually. In this mode of play, two or more people make, refute, strengthen, and agree with claims all via pen and paper without the assistance of the SCG software.

In this paper I will analyze what I view as the strengths and weaknesses of the SCG system, both in manual play and software driven play. In addition, I will discuss what I view as several key improvements that are necessary in order for the SCG software to be useful to the broader public.

SCG Scholar

SCG Scholar is the manual form of the SCG playground. In this version of the game, players manually compare their algorithms against one another through pen and paper. In practice, I found this method of developing algorithms time consuming and largely unhelpful. Usually the only people I had available to play with were people with whom I'd already discussed what the best solution might be. This meant that we were pitting the same algorithmic claims against one another, and were unable to evolve our strategies.

Even if I had broadened my search and played games against a larger pool of participants, I suspect the outcomes would still be of limited practical use. I say this because often it's not enough to discuss the theory of an algorithm when the devil is in the details of the implementation. For example one implementation of an algorithm that uses an ArrayList might run in $O(n^2)$ time whereas another implementation of the same algorithm that uses a Stack could run in $O(n)$ time. There's no way to see these details without actually implementing the algorithm.

SCG Avatar

The SCG Avatar game, which uses the SCG software to drive the tournaments, is orders of magnitude more powerful than its manual based sibling. The software allows many avatars of differing abilities and origins to compete with one another. This grows the playing field quickly and easily and makes for easy analysis of a large number of claims over the course of a tournament.

Another advantage of using the SCG software is that it allows you to implement your algorithm under the circumstances that you're attempting to solve it with. That means that rather than your brain producing a solution, you have to create data models and manipulate them in the most efficient way to allow your program to produce an optimal solution. Once perfected through tournaments in SCG, you can easily transfer your solution into whatever context you actually need it in.

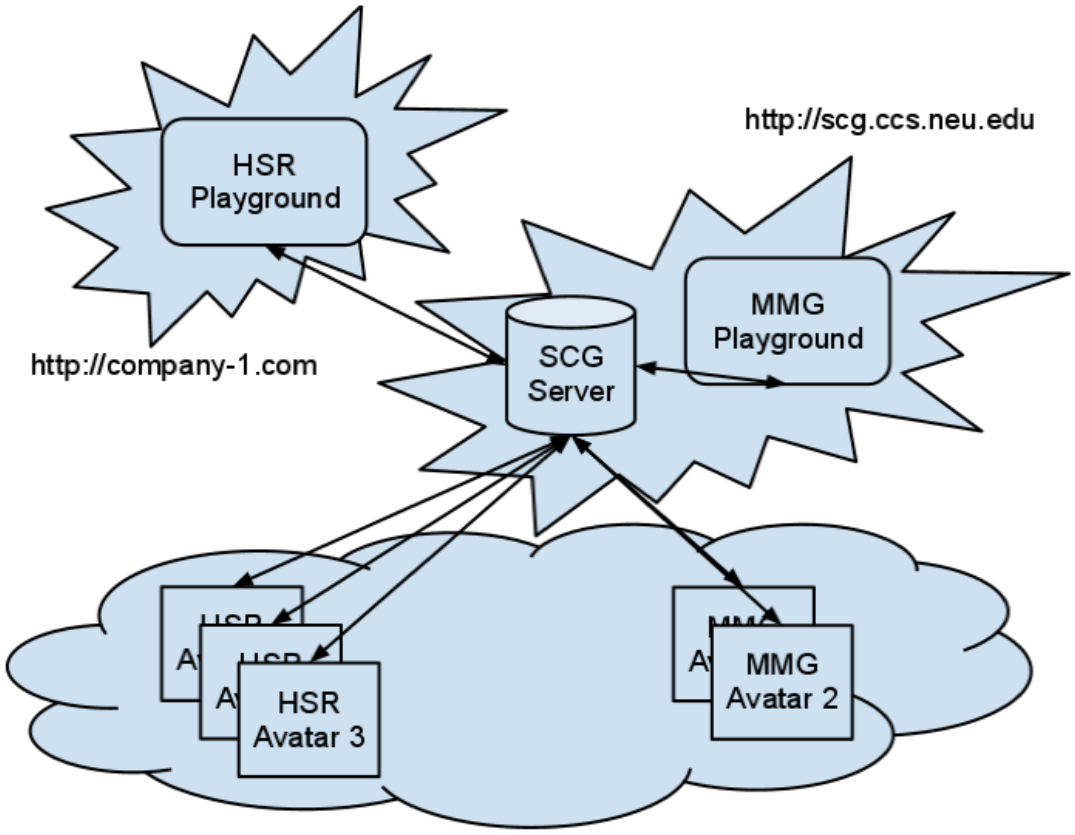
The Utility of SCG

Bugs in the software implementation aside, SCG still has a way to go before it can truly be useful to outside consumers. One of the primary difficulties with its current implementation is that it relies heavily on Java. Although it is be possible to implement a non-Java based avatar, it would be a lot of work. This could be quite easily avoided by changing the communication protocols to use a standard format, such as JSON or XML. Since most languages have libraries available to consume and produce JSON or XML, it would be far easier to implement an avatar in whatever language you desire. With this small change, people who want to run a tournament to find the quickest implementation of HSR using Python could do that. Or if they wanted to compare a Python implementation with a Ruby implementation, they could do that.

Another major drawback of the current iteration of the product is that the client seems very strongly coupled with the server. We had several tournaments where the server software was running a new version and the grad students told us that we might be able to run the old version of the clients against it, but they weren't sure. Forcing clients to upgrade unnecessarily can be difficult and unpopular for various reasons. The ideal solution would be to come up with an agreed upon communication protocol (as described above) and then split the client code into a completely different distribution from the server code. That way, the only time you'll need your clients to upgrade is when there's a change in the communication protocol, which should almost never happen.

Once a standardized communication protocol has been implemented and the client code has been split from the server code, there is one other logical change that would make distribution and use of SCG easier. If you split the playground implementation out from the server implementation, you can have users easily implement and hotswap playgrounds. This could work much in the same way as users can implement and hotswap avatars at the moment. This final change will allow a single hosted version of SCG to be running multiple dynamic playgrounds at

a time and give users the ability to implement a playground in whatever language they desire.



I've tried to illustrate this concept in the above diagram. In this instance the college is running a single hosted version of the SCG server. The college is also providing a playground for the MMG problem. In addition, an external company has registered a playground that they want people to compete in. This playground could be implemented in Java, Python, Ruby, C#, etc, it doesn't mater as long as they perform valid and quality checks correctly. The company and the college can then advertise their respective tournaments however they want and people from all over the world are able to enter their avatars in those tournaments.

The advantage of this distributed architecture is that playground implementers don't necessarily have to make any changes if the SCG server needs to be upgraded. They can also easily maintain and update their playground without having to recompile it against the central server. Finally, it means that if there's a playground for which it's difficult to compute the validity of a solution, it won't impact any other tournaments since the computations will be running on a remote machine.

I would still recommend distributing the SCG server executable so that people can host their own private SCG servers. However, this architecture would lower the bar of entry for people wishing to host a tournament because they could easily host it on the constantly running SCG server run by the college (or whoever maintains it).

Conclusion

I strongly believe that the SCG concept is an extremely powerful and appealing idea. I can also say that it absolutely helped me when developing my implementations of both the HSR algorithm and the MMG algorithm. However, I believe that the implementation of the software still needs some adjustments and a little polish before it will be easily usable and helpful to the outside world.