Project Proposal
Alex Gajewski (apg2162)

For my project, I'm planning on making a simple open-ended ALife simulator. Open-ended is a keyword here. Most ALife simulators use an explicit evolutionary process, with organisms that have some genetic code and some reproduction mechanism. They also usually have some sense of evolutionary fitness that the organisms are trying to optimize. My proposal is to do away all these things and simulate at a slightly lower level, where you just have little bits of code that can modify each other and themselves, and a bit of randomness thrown in. The idea is that something like reproduction would probably emerge, because if any such idea happened to emerge by chance, it would quickly expand and take over everything. The only question is how long it would take until such a thing emerged.

More concretely, the idea is this: there is a list of code expressions (imagine Lisp S-expressions for something specific) that represents the universe, and there is some fixed number of threads of execution, representing energy in this universe (this is where parallelization could give a speedup). Each thread of execution is always evaluating some expression in the list, and if it ever finishes, it jumps to another random one and starts up again. There is also some base rate of mutation, by which randomly selected code expressions are mutated according to some mutation rule. Importantly, some of the instructions allow the code expressions to read and write their own and each others' code. This should be done through relative addressing so that the same expression can be used for recursion anywhere in the list. For example, an expression should ask for (expr 0) to get itself, (expr 1) for its neighbor to the right, (expr -1) for its neighbor to the left, etc. This mechanism, together with the ability to pass execution to one of these other expressions (e.g. (call (expr -1))) ought to be enough to make the whole thing Turing-complete. In particular, in the simplest implementation, you don't even need a concept of a variable or a function--just this top-level universe. At the beginning of the simulation, the universe list will be filled with randomly generated code expressions, and the threads of execution will be assigned to randomly selected expressions. At this point, the game will be to see how long it takes for recursion to occur (all you need is one of the expressions to be (call (expr 0)) to capture the execution thread indefinitely, until a mutation kills the program), and to try to study any other patterns that emerge.

The implementation for this simulator should be fairly straightforward, but lots of fun. One part will be building a tiny interpreter for the language the code expressions will be in. Another part will be doing the actual simulation, whereby threads of execution jump around the list of expressions and mutations modify its contents. For the parallelization component of the project, I'll try to speed up the instructions-per-second of the simulation by parallelizing the threads of execution. If I have extra time, I might try to find a way of visualizing the simulator while it's running, or of saving a run to disk and visualizing the results after the fact.