PIPLUP:
Programmatic Inference of Poetic Language Underlying Processes

Goal

A natural language processing based deconstruction of poetic passages. Ideally the system will be trained on a wide variety of poem and song sample texts. Learning from the sample data, it should generate heuristics for constructing novel poetic texts of its own.

Methods/Design

The first, and most important component of this system is a thorough linguistic package which can extrapolate the essential syntactic, semantic, pragmatic, morphological, and phonological characteristics of the poetic texts from which it is to learn. The methods contained in this package will, when used on a large body of sample texts, generate a probabilistic context-free grammar (PCFG) from gathered statistics about the creative structures and linguistic components of the texts.

Secondly, there will be a system which uses the PCFG production rules and some dynamically generated heuristics to generate novel, (hopefully) poetic texts, subject to optional constraints given by the user.

Separating "form" and "meaning"

As the field of linguistics makes such a distinction between form and meaning, it is only appropriate that this system do so as well. LISP, which is a language that prides itself on having the unique property of representing data and code in the same way, should provide some interesting approaches to dealing with this problem.

One aspect of the deconstruction process will be dealing with literal forms (syntax, morphology, phonology) and generating the PCFG, while another will be working purely in the linguistic notions of semantics and pragmatics in order to generate heuristics which will in turn generate meaningful relationships of literal forms. These generated heuristics will be able to provide the text generation algorithm with constraints (generally, allowing only those grammar productions that conform to given meaningful relationship(s)).

Resources

On top of the application code, a large knowledge base will be required of this program. This knowledge base will consist of at least two components: A linguistic "dictionary" which contains words and annotated linguistic information on them, and poetic text corpus, which will just be a collection of short poems/songs in a single plain text file. After reading a text from the corpus, the application code will utilize the linguistic information from the "dictionary" to create statistical constructs for observed patterns.

PHENoLimit:
Poker Heuristic Expresser - No Limit

Goal

An AI system for a no-limit hold'em poker game that generates heuristics based on observed behaviors of its human opponent. The most important aspect is that the program does not know what the human's hand is (unless revealed at the end of a play, of course).

Methods/Design

A key feature of any good poker player is the ability to learn not only from one's mistakes, but also from the successes and failures of an opponent. This system will attempt to take that philosophy and apply it programmatically through automatic heuristic generation. These set of generated heuristics will be based on information gleaned from each play, and will consist of a mix of specific situational rules based on particular plays and statistical rules derived from patterns across multiple plays.

A good poker AI needs a good poker environment, and that will be the first stage of this project: a solid, full-featured poker game. As with the Quarto project in csc416, this game will be built from the ground up, so as to be able to incrementally implement the AI of the machine player. Once the heuristic machine player is implemented, it will start out playing (betting, checking, folding) randomly, and gradually increase it's knowledge of the game only through interactions with its human opponent (and its base knowledge of how the game is played, i.e. what constitutes a win, loss, or draw).

Heuristic generation will be based on a number of factors, including, but not limited to: opponent's hand (revealed/not revealed?), success/failure, betting/checking/folding patterns, flop/river/turn patterns, own hand (in relation to opponent's if revealed), or any combination therein.

Interface

Initially, a command line interface will suffice, although if time permits, I would like to take this project as an opportunity to learn to use some graphics library(s) with LISP.