Implementation Instructions

There are two files which must be implemented.

1. FitnessFunction.java
2. GAIndividual.java

You need an Individual class, and a fitness function. As long as the files follow the interfaces, there is no other preparation necessary.

We have provided examples to help explain the implementation process.

Checkers Implementation

Patrick Burke

As a true test of our Genetic Algorithm Framework, we decided to develop an implementation of a checkers player. Using a version of Minimax[1] copyrighted in 1997 by Victor Huang and Sung Ha Huh. The actual element of the checkers player that evolves is the heuristic function, which decides the value of a board at any given point in the game.

How Minimax Works

Minimax does an exhaustive search, taking on the role of itself, as well as its adversary. It finds either the maximal move for itself, or the move of minimal gain, when playing for its opponent, assuming that its opponent will always take the best move. However, Minimax cannot search until the end of the game because of the time it would take to evaluate such an enormous number of moves. Instead, it searches to a specified depth, and then does an evaluation of the last position.

Heuristic Function

We have focused on creating the function that evaluates the position at minimax's final depth. This function makes the biggest difference in the overall performance of the algorithm. Although the heuristic function could be used without minimax, it is necessary to use them together to ensure the highest effectiveness.

What We Did

We created an implementation of GAIndividual called CheckersEvaluator. This class implements GAIndividual, which means that it provides the Framework with the functionality of reproducing, and creating random instances of itself. This is not enough to be effective however. This class also implements the Evaluator interface within the Checkers Implementation, in order for the individual to be plugged into a game as an evaluator. The current evaluator takes a series of statistics from the board, and applies certain weights to these statistics, on a scale of -10 to +10. The total evaluation of the board is based on these values.

The attributes themselves are kept in a list of numbers in the scale. These numbers are unique to each individual. We used the genetic algorithm to decide which of these individuals would "reproduce" by sharing some of its numbers with another.

Our fitness functions are tournament based, which means that each individual plays a series of games against other players. Our beginning function plays against a default evaluator, but after a period of time, we found that it was necessary to play the individuals against themselves, in order to continue to improve the quality of our players.

It's So Easy

The actual implementation of the Checkers player took very little time. Two classes needed to be created, and evolution began. These classes are very simple in format, and can be created quickly. The individual and fitness function classes could then be perfected based on our evaluation of their strengths and weaknesses.

Other Implementations

We have created various other implementations, such as tic-tac-toe, 16 bit strings, and other simple problems.

---

[1] Minimax is an adversarial search algorithm used often in artificial intelligence.

Bibliography

1. Davis, L Handbook of Genetic Algorithms 1991. New York, NY. Van Nostrand Reinhold.

2. Goldberg, D Genetic Algorithms in Search, Optimization, and Machine Learning. 1989. Reading, Mass. Addison-Wesley Publishing Co.