Connect-K Project Specification

GENERAL:
You are employed by a company that makes interactive video games. You have been provided with a "dumb" GUI shell and assigned to code the "smarts." Your product should (1) make each move in no more than five seconds (which should be an adjustable parameter in your code --- never hard-code anything!); and (2) beat you at Connect-K. You must form a project team of two people (follow the "Pair Programming Paradigm," switch roles often; /en.wikipedia.org/wiki/Pair_programming).

- Where S is a game state, you must code a static heuristic evaluation function, Eval(S), which estimates how good is the current board configuration S. Eval(S) is usually the dot product of a vector of weights with a vector of feature values, as discussed in the lecture slides.
- Implement Alpha-Beta pruning in your search of the game tree (put it on a switch so that you can turn it on and off, and thereby evaluate how much it helps you).
- Implement Iterative Deepening Search. When the clock runs out you return the solution found at the previous depth limit.
- By remembering the values associated with each node in the game tree at the previous depth limit, you can sort the children at each node of the current iteration so that the best values for each player are (usually) found first, and so alpha-beta pruning will become (almost) optimally efficient.
- It certainly would seem foolish to cut off search in a game state where the opponent could win the game in the very next move. The quiescence test checks for this condition. If true, it allows the search to be extended for one more move. The quiescence test is then applied again.

CODING:
Your code must run on the ICS lab machines. You may develop it on any platform you like. Please start early. All coding takes longer and is more complicated than initially anticipated. It is far better to have code that is simple and working than to have code that is complicated and unfinished.

Eval(S).

The Possible Win Paths heuristic was discussed in the lecture slides (Game Search; Adversarial Search). A possible win path is any possible extension of the current board state that results in a win for the player. The possible win path evaluation counts the number of possible win paths for each player, then subtracts one from the other.

A better solution is the dot product of a vector of weights with a vector of features, as discussed in lecture. The Possible Win Paths above would be one feature, and your insight and creativity would provide others. For example, multiple intersecting win paths, win paths near the center of activity, win paths near the center of activity, ability to block possible win paths, etc., as limited only by your won creativity in thinking about the problem. Set the weight vector empirically to maximize performance, either by playing you (slow) or by playing itself with different weight vectors (faster).

Quiescence(S) evaluates whether or not a quiescent position has been reached, and if not, extends the search one more level (recursively). This guards against the "Horizon Effect" (see section 5.4.2). Your quiescence test should at least check to see if the game can be won in the next move; your creativity may suggest other cases as well.