General Recursive Algorithm

```
/** Append to PATH a sequence of knight moves starting at ROW, COL
 *  that avoids all squares that have been hit already and
 *  that ends up one square away from ENDROW, ENDCOL. B[i][j] is
 *  true iff row i and column j have been hit on PATH so far.
 *  Returns true if it succeeds, else false (with no change to PATH).
 *  Call initially with PATH containing the starting square, and
 *  the starting square (only) marked in B. */

boolean findPath (boolean[][] b, int row, int col,
                  int endRow, int endCol, List path) {
  if (path.size () == 64)   return isKnightMove (row, col, endRow, endCol);
  for (r, c = all possible moves from (row, col)) {
    if (! b[r][c]) {
      b[r][c] = true; // Mark the square
      path.add (new Move (r, c));
      if (findPath (b, r, c, endRow, endCol, path))  return true;
      b[r][c] = false; // Backtrack out of the move.
      path.remove (path.size ()-1);
    }
  }
  return false;
}
```

Some Pseudocode for Searching

```
/** A legal move for WHO that either has an estimated value >= CUTOFF
 *  or that has the best estimated value for player WHO, starting from
 *  position START, and looking up to DEPTH moves ahead.   */
Move findBestMove (Player who, Position start, int depth, double cutoff)
{
  if (start is a won position for who) return WON_GAME; /* Value ∞ */
  else if (start is a lost position for who) return LOST_GAME; /* Value −∞ */
  else if (depth == 0) return guessBestMove (who, start, cutoff);

  Move bestSoFar = REALLY_BAD_MOVE;
  for (each legal move, M, for who from position start) {
    Position next = start.makeMove (M);
    Move response =  findBestMove (who.opponent (), next,
                                   depth-1, -bestSoFar.value ());
    if (-response.value () > bestSoFar.value ()) {
      Set M's value to -response.value (); // Value for who = - Value for opponent
      bestSoFar = M;
      if (M.value () >= cutoff) break;
    }
  }
  return bestSoFar;
}
```

Static Evaluation

- This leaves static evaluation, which looks just at the next possible move:

```
Move guessBestMove (Player who, Position start, double cutoff)
{
  Move bestSoFar;
  bestSoFar = Move.REALLY_BAD_MOVE;
  for (each legal move, M, for who from position start) {
    Position next = start.makeMove (M);
    Set M's value to heuristic guess of value to who of next;
    if (M.value () > bestSoFar.value ()) {
      bestSoFar = M;
      if (M.value () >= cutoff)
        break;
    }
  }
  return bestSoFar;
}
```