Homework 3
Computer Science
B351 Spring 2017
Prof. M.M. Dalkilic

James Gregory

February 27, 2017

All the work herein is mine.

Introduction

The aim of this homework is to get you well-acquainted with $\alpha\beta$. You will turn-in two files

- A *pdf with the written answers called `h3.pdf`
- A Python script called `gobblet.py`

. If you've attempted extra credit, add the comment #ExtraCredit to the programs and ExtraCredit to the homework near the top so it's visible and obvious. Please enjoy this homework and ask yourself what interests you and then how can you add that interest to it! Finally, problems 1 and 2 are worth 100 each and problem 3 is worth 300 points. Include the statement, "All the work herein is mine."

Homework Questions

1. A *general search strategy* is to work from both the start and goal–think of navigating a maze. Is this a sound strategy for a game? Recall that soundness means $\vdash_{Robot} A \rightarrow \models_{Human} A$

   As long as care is taken to ensure that each path is an optimal path, and the insterecting point of the two paths chosen is the optimal intersection; yes this strategy would be sound for a game. This is assuming the goal-state of the game is known beforehand (as in 15-puzzle or Nim). If it is not required to be optimally sound, then bidirectional search is always sound. Again, this assumes that the bi-directional search is applicable to the game (i.e. goal-state is known and both predecessors and succecors are generatable from any given node).

2. Assume there is a game with three players A, B, C. You have an evaluation function h that returns 3 numbers: $h(\sigma) = \langle A_s, B_s, C_s \rangle$ where σ is a state of the game and the numbers reflect the goodness of the state for the respectively named players. In no more than two paragraphs, is there a way to modify minimax to work with 3 players? Assume that you *cannot* exhaustively search the game space, but you're able to generate, from a state, all the next states.

   Regardless of the number of players, most games can still be seen as a zero sum game. By this, I mean the player we're trying to choose the best move for either wins or loses. It doesn't matter which of the other players wins, if it's not our selected player, then we lose. This means that minimax would essentially work the same with one simple change. Instead of alternating turn order as $max \rightarrow min \rightarrow max...$, we would do it as $max \rightarrow min \rightarrow min \rightarrow max....$ Looking at the turn order in this way, we would perculate scores up the same way as 2-player minimax, using the score of the player we are selecting a move for. Then we would choose the action that results in the highest h-value available to our player.

3. Implement $\alpha\beta$ for the game of gobblet. Assume that if a state σ is repeated, then the game is a draw:

   $$\forall(\sigma)[\mathsf{move}(\mathsf{move}(\sigma, player1), player2) = \sigma] \rightarrow (\mathsf{game}(\sigma) := \mathsf{draw}$$

   The game should be able to play: human vs. human = h2, human vs. robot = hr (human goes first), robot vs. human = rh (robot goes first), robot vs. robot = r2. There are two sets of parameters:

   (a) Level: beginer = 0, intermediate = 1, expert = 2. This places a limit on the depth of the search. You must decide experimentally how this is applied.

   (b) Time: x min. This is the bound on the time you run $\alpha\beta$.

   The main function should be called $\mathsf{gobby}(players, level, time)$ where $players \in \{$h2, hr, rh, r2$\}$. The output should be a sequence of moves, with the final state when it's won or drawn. $\mathsf{gobby}(\mathsf{r2}, 2, 2.5)$ means robot versus robot, expert, no longer than two minutes and 30 seconds should elapse *after* the opponent makes a move. Whatever is unspecified at this point, you must make decisions on.

   see gobblet.py