Intro to AI End of Term Assignment

Andrew Hynes, Samuel Sleight, Helen Cheung, Amar Saggu

2013-12-13

Contents

1 Overview of the Program

The basic game structure is that the game is represented in an integer array. Positions 0 - 5 represent the top row of houses, position 6 represents the first player's store. Positions 7-12 represent the bottom row of houses, and position 13 represents the second player's store.

We had both of our AI extend a class called AIBase, which get passed into the playGame method when they play the game. The AI individually have a method called makeMove, which will implement their algoririthm and will call the "sow" method of the KalahGame class, which will in turn modify the state of the board, ready for the next AI's move.

After constructing the infrastructure of the game and whatnot, including the base AI class and all of the plumbing for making sure everything connects properly, we went off into our subgroups and made out individual AI which extend the abstract AIBase class. Needless to say, the methods (such as playGame) take an object of type AIBase, which meant we could do all of the game together before we'd made our AI, utilising abstract classes.

We used GitHub as version control. The repository can be found here - `https://github.com/YeyaSwizaw/kalah`. We found this was the easiest way to collaborate as a group to create coherent code that meshed well with each other, even despite it not being coded all by one person. This is a lot more efficient and sensible than simply having a dropbox folder, we found, as it's less fiddling around.

The results of our 1000 games were -

AI 1 - Helen & Amar's - X/1000 wins

AI 2 - Sam & Andrew's - X/1000 wins

2 TODO Something Algorithm - By Helen and Amar

Put stuff here!

3 MASH Algorithm - By Sam and Andrew

This is the algorithm and AI constructed by Sam and Andrew, which can be found in MASH.java.

3.1 Basis of the Algorithm

We based our algorithm largely on the M&N algorithm - an improement on the mini-max algorithm. We chose this as it has been greatly successful in the past, and an AI written in Lisp utilising this algorithm has won tournaments with other AI based on other algorithms before. In short, the M&N algorithm has been found to perform significantly better than a base mini-max algorithm.

We found a PDF on the M&N algorithm here - `http://dl.acm.org/citation.cfm?id=362054` and though it was originally written in Common Lisp, we took the ideas of the M&N algorithm, namely that a min-max algorithm should pick from a few options and take into account relative uncertainty (especially considering the fact that algorithms for this task are designed to learn) - therefore we can't be certain as to whether the opposing AI will modify their moves using what they've learnt (potentially from how our AI plays) from the last game(s).

We also took some inspiration from Artificial Intelligence: A Modern Approach, for example, pages 480 - 483, and applied its comments on reasoning under uncertainty to our implementation of the M&N algorithm. We felt it

would be prudent, when against any decent learning algorithm, to consider uncertainty when we are unsure, indeed, what move the opposing AI will choose, and whether they will have adapted their efforts from last time. The book proved useful a great deal for referencing in regards to how to construct a sensible AI, and gave us some places to start with algorithms and design.

3.2 Analysis of Behaviour

3.2.1 Expectations

We expected our algorithm to perform quite well throughout the 1000 games. We expected the learning we utilised to not gain a giant lead from the other AI, rather, to mainly quash and keep up with the opposing team's efforts of learning from our AI. Rather than having a huge boost in improvement as time went on, we expected a slight boost, but that would also be counteracted by the fact the opposing AI was also learning.

We expected our AI's lead (if one existed) to stay relatively constant as time went on, and any growth or reduction in performance to be slight. Our algorithm didn't start out entirely naively and learn rapidly - it utilised search as well as learning to get a nice foothold immediately.

3.2.2 Performance

Our algorithm performed