Artificial Intelligence
2019
Assignment 2
Connect Four

S3602584 Peter Moorhead
S3603837 Tim Dalzotto

The main algorithm behind the AI that we created is the minimax algorithm, although there was the option to use other techniques, we kept with it because of its usefulness when calculating the best move in two player games such at Connect Four. This works by creating a simulation of all possible games of Connect Four and represents them as a tree. The root of the tree will be represented as the empty board. The second level of the tree will be represented as the possible moves that the first player can make. The third level of the tree will be represented by all the possible moves that could be made by the second, and so on. Each level of the tree is represented in terms of a loss or gain for our AI. In order for our AI to make moves using the minimax algorithm, the AI assumes that its opponent will always play optimally, and our AI will try to minimize the opponents gain. According to 'the online encyclopedia of integer sequences' (source: http://oeis.org/A212693) there are over 4 trillion possible games of Connect Four and thus it would be impossible to generate every game within a reasonable time. However with the inclusion of Alpha Beta pruning it allows the reduction of the number of nodes needed to evaluate from 4 trillion to around 2.7 billion boards. Alpha Beta pruning stops once a board is found to be worse than the previously examined board.

Our evaluation function iterates through the current board state and adds or takes away from a variable boolean 'playerOne' which we return to tell the minimax agent whether we are in a winning position or a losing position. It looks for a streak of tokens either for or against the StudentAgent, 1 2 or 3 in a row, + or - 10, 100, 10000 respectively depending on how good or severe the number of tokens there are in a line and how many lines there are. We chose this initial evaluation because from our early research we found that iterating through the board is the most widespread technique and because it was a simple base where we could build a more advanced function.

We then modified the minimax function and the max depth of the agent to search 4 layers deep so that it can detect for a few different scenarios. Firstly, if searches at depth 2 for the case that the StudentAgent will win at that point, and gives a large positive priority so that the AI will expand upon that strategy. Secondly it does the same except searches through 2 depth for the opponent winning, and then gives a large negative priority so that the Ai will move to block the opponent from winning. Getting the AI to block when needed but not block instead of winning required some amount of tweaking the priority numbers given. Lastly, the

AI detects at 3 and 4 depth, giving less negative priority to these levels, progressively as they are less important than the next turn or the turn after.

We mainly tested against the MonteCarloAgent as we thought this would give us a better idea of where we stood as we had progressed past learning anything from beating the RandomAgent over and over again. Through testing against Monte Carlo we learned that in some niche cases, our AI will prioritise blocking the opponent when there is an opportunity to win, although as this happens in the minority of cases we couldn't track down the exact cause of the issue. In the beginning when developing our evaluation function, through our testing, we saw that going first our AI performed much better than going second. Unsure of the cause we researched further AI strategies and then implemented a search that would evaluate through further depths so that it could detect the opponents moves 2 to 4 steps ahead. This helped give our AI some more parameters to work with, this thus lead to our code working much better and improving our winrate going second against the MonteCarloAgent quite a lot. With further testing we find that we win almost every time going second and fractionally less than that going second. As previously stated Alpha Beta pruning would be used to reduce the amount of nodes that would need to be evaluated allowing the AI to search to a greater depth. Since we were not able to successfully implement Alpha Beta pruning our AI was limited as to the depth in which it could search within the given time-limit. This forced our AI to make the most optimal move available within its given depth, although this sometimes proved suboptimal.