CS165A MP2

Brian Lee

March 2017

1 Architecture

For this project, I have three classes in addition to my main class: an AI class, a Board class, and a Game class. My main class is the simplest: it takes in the command line arguments and uses them to create a game that matches the user's specifications. The Board class holds the game's board, keeping track of where the pieces are and determines the heuristics of the current state. My AI class determines the best possible move that the computer can make through a minimax search with alpha-beta pruning. And finally, my Game class holds all of the functions that require the game to work.

The Game class is the simplest. It really only has one member function not related to the initialization of the class, but it has a few helper functions designed to facilitate the game. It takes in the string input and changes it into a pair of ints that can be fed into the Board class. It keeps track of who the current player is, and facilitates the game for the human player. And it holds a few end-game messages to give it that human touch, along with an exit case for when there is no way for either player to win.

My AI class is more complex. It needs to be able to determine the heuristics of a move and search through it through a minimax search. The AI does this through a recursive function that creates a vector of structs called positions and returns the best one.

Finally, there is my Board class, which happens to be my most complicated class. It keeps track of the pieces on the board through the use of a 2D vector of ints. If the integer is -1, the position holds a Dark piece. If the integer is -2, the piece is light. For all other values, the position holds nothing. In addition, the Board class is also responsible for printing out the board itself. However, there is one more important thing that the board class does, and that is evaluate and hold the heuristics of the current state. This is determined by a few factors, namely how much room the position has around it, the possibility of winning the game, and how many of the player's pieces are around it.

2 Search

As mentioned earlier, I used a recursive function to effectively simulate a minimax search. The function keeps track of whether or not it's at a min or a max level, and it keeps track of both its maximum breadth and depth. Once it reaches the maximum depth, it selects the state with the highest heuristic value, which is determined by the board class.

The function works by creating a copy of the current board and adding the move that the search was started with. Once it does that, it makes a vector of possible moves and runs through it, calling itself recursively for each move in the list. If the value of the move is better than the value that was set in the function, the move replaces the value. The function returns the best possible value, which lets the ai know which move is the best one to go with.

3 Challenges

My main challenges was in implementing the minimax search. The problem was that, though I understood the theory, I had no idea how to implement it without using a data structure. That is, until I hit upon the idea of running through the search through a recursive function.

Once I had figured that out, my other challenge was in determining the heuristics of the board. How could you quantify, in numbers, how well a game was going? I decided to sum up of the positions of the board, determined by factors like their position relative to other pieces, for each player. In other words, each board had a heuristic value for each player, not to mention that each position will also have its own heuristic. So, to keep track of these player specific heuristics, I have two other vectors besides the game board itself, which holds the same info as the board, but also has the heuristic values for each player.

4 Weaknesses

There are two inherent weaknesses to my method. The first is in determining heuristics. It is entirely possible, if not probable, that my way of determining the heuristics of the board is not the best measure. If there are better measures, then my AI will likely lose out to them.

The second weakness is in my search itself. Because of how I limited my search, it is possible that I'm missing out on better moves or better strategies. In addition, since my depth is shallow for time reasons, the strategy is not as deep as it could go. In other words, my AI can play a frustrating short-game, but it has little strategic vision. Granted, the recursion does speed up the process greatly, but because of the heuristics, it is not likely to improve anytime soon.