Chess AI

Seyed Mahdi Basiri Azad (Erfan)

February 10, 2016

1 Minimax Algorithm

Before starting to implement the iterative deepening Alpha-Beta search for the computer, I implemented the minimax algorithm. For that reason I started by implementing the **cutoff, utility, and evaluation** functions. They are listed below:

```
 //Checks if the current position is terminal or the maximum depth has exceeded
 private boolean isCutOff(Position position, int currentDepth, int maxDepth){
return (position.isTerminal() || (currentDepth > maxDepth));
 }
```

```
//return an int representing the value of the given position
private int utility(Position position){
 if(!position.isTerminal()){//if not a terminal position, estimate
  return eval(position);
 }else{
  if(position.isStaleMate()){ //the this position is a draw
return 0;
  }else{// if it is a mate && its the AI's turn, bad for AI, good for the other
return (position.getToPlay() == AIplayer) ? MinVal : MaxVal;
  }
 }
}
```

And I calculate the Estimation using the eval function:

```
//Estimates a value for the given position
private int eval(Position position){
 int index = (int)(position.getHashCode())%MAXHASHSIZE;
 if(TT.containsKey(index)) { //if the value is already in the hashtable return it
  return TT.get(index);
 }else{
 //calculate the position value
  int posValue = position.getMaterial() + (int)(position.getDomination());
  //adjust for AI or other player
  posValue = (position.getToPlay() == AIplayer) ? posValue : -posValue;
  TT.put(index, posValue); //add to the transposition table
  return posValue;
 }
}
```

2 Minimax vs. AlphaBeta

Using minimax I could get to a depth of 4 within a reasonable time (1 min maximum) after changing to AlphaBeta pruning I could increase the depth to 6 within that time. below is the comparison between **Iterative-deepening-minimax** and **Iterative-deepening-AlphaBeta** in terms of the number of states explored:

ID-minimax (at depth 4): 9,642,768

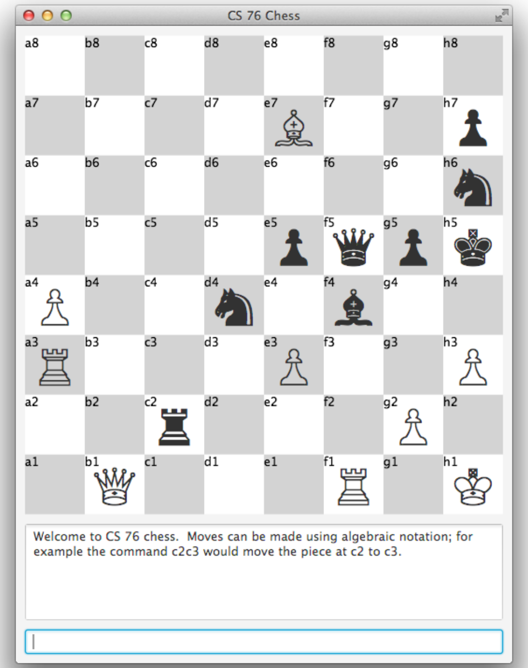
ID-AlphaBeta (at depth 4): 365,270

As it is shown AlphaBeta pruning proved very successful for optimizing the minimax algorithm.

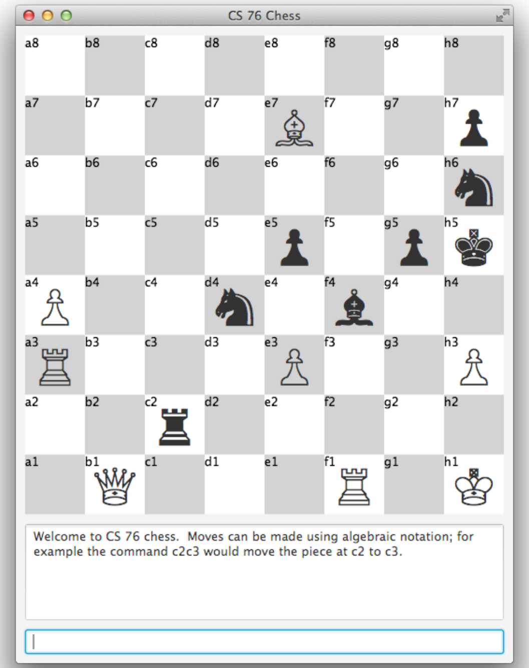
3 Transposition Table

I spend a majority of this project trying to implement the transposition table and optimize the code. Unfortunately some last minute changes broke the code however I do have pictures of one of the FEN sequences from piazza from the time that it was working (I was not using the transposition table at that point!).

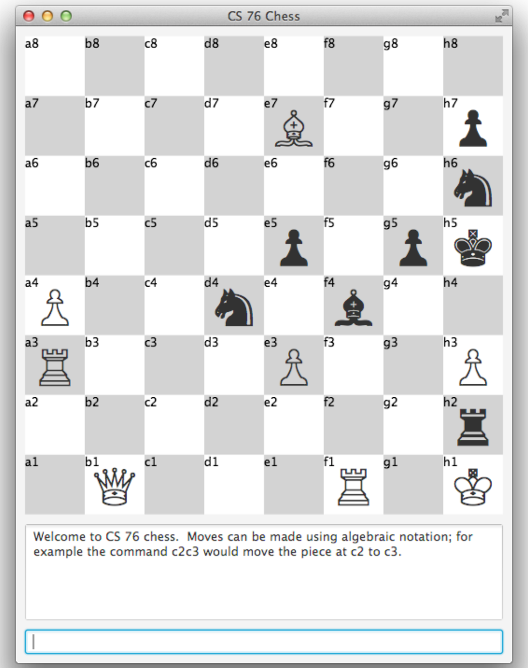
Black (AI) is starting by capturing the white pawn with its queen...



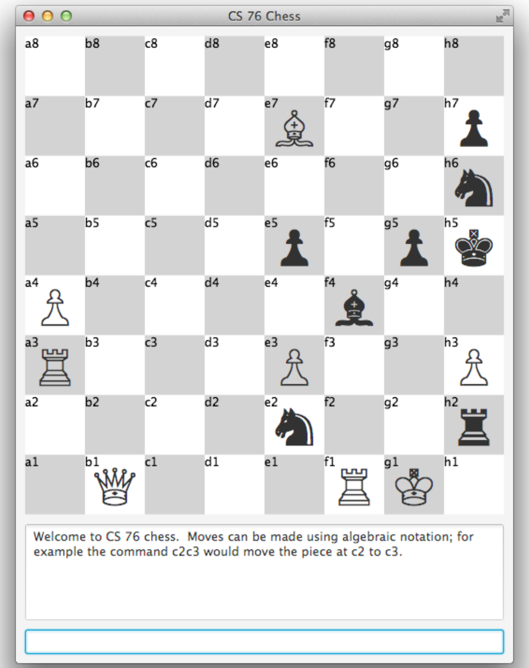
Welcome to CS 76 chess. Moves can be made using algebraic notation; for example the command c2c3 would move the piece at c2 to c3.



Welcome to CS 76 chess. Moves can be made using algebraic notation; for example the command c2c3 would move the piece at c2 to c3.



Welcome to CS 76 chess. Moves can be made using algebraic notation; for example the command c2c3 would move the piece at c2 to c3.



Welcome to CS 76 chess. Moves can be made using algebraic notation; for example the command c2c3 would move the piece at c2 to c3.

In the beginning I created a private ”TTEntry” class to hold information about the position and save that in the transposition table. Below is the code for the TTEntry class:

```
private class TTEntry{
    Long zobrist; //From position.getHashCode()
    int depth; //Depth in which this position was found
    int utility; //Estimated value of this position at depth above

    public TTEntry(Long z, int d, int u){
        zobrist = z;
        depth = d;
        utility = u;
    }
}
```

I defined the transposition table as:

```
//Used to initialize the Transportation table size
public final static int INITIALHASHSIZE = 1000;
//Used to limit the size of the transposition table
public final static int MAXHASHSIZE = 10000;
public LinkedHashMap<Integer, TTEntry> TT =
    new LinkedHashMap<Integer, TTEntry>(INITIALHASHSIZE, 0.75f, true){
        protected boolean removeEldestEntry(Map.Entry eldest){
            return size() > MAXHASHSIZE;
        }};
```

I followed the tutorial from `http://mediocrechess.blogspot.com/2007/01/guide-transposition-tables.html` to create the transposition table at first. After trying to use the table I realized that the table was helping as long as I kept the maximum size of the table below 10,000 (experimentally determined!). Below are my results of number of positions visited at depth 4 using Iterative-Deepening-AlphaBeta with and without transposition table (of 10,000 maximum size):

without TT (at depth 4): 2,271,969

with TT (at depth 4): 1,733,043

From the comparison above we can see that the transposition table helped by reducing the need to visit the positions that are already visited (at least some of them!). The problem with it was that despite the reduction in the number of visited positions the overhead from the table would make the program run the same if not worse in terms of time. That was the main reason for my en devour to optimize the code and the table. I started by getting rid of the TTEntry class and replacing it with the estimated value of the position (unlike what was suggested in `http://mediocrechess.blogspot.com/2007/01/guide-transposition-tables.html`). Next I tried to improve the AlphaBeta by looking at the capturing moves before the non capturing ones. This is the part that I have made the most changes to my code and unfortunately it is not running at the moment so I cannot comment on the impact of these changes on the performance! (You can say I kinda Fez-ed it! `https://en.wikipedia.org/wiki/Indie_Game:_The_Movie`)

There is a "Unused Code" section at the end of AlphaAI.java file that contains the codes related to the transposition table and the minimax algorithms before the changes!