COMP30260 - Assignment 2

Fidchell with Transposition-Table Enhanced Negamax Alpha-Beta

Kevin Heffernan
ID: 11277955
kevin.heffernan@ucdconnect.ie

November 29, 2013

1 Note to Arthur

For this second homework assignment, I have written my source in python, and this accompanying document in LaTeX (please email me if you need the .TeX file). Also, sorry this report is a little late, scaling down my board from 9x9 to 7x7 and trying to undo user move take-back proved non-trivial. I kept in user move take-back in the end, so a user is prompted at the end of each move, whether or not they would like to make a take-back which then pops off the number of requested moves from the game-state stack, and also resets the Zobrist Key to that at the time of the requested past move.

2 Static Evaluation Function

Since my evaluation function is going to be called many times, I tried to make it simple and efficient, yet as effective as possible. It works as follows:

If the High King has been captured or has made it to the corner square, return {+/−} 1 million (depending on whose turn it is to move)

If those two conditions have not been met, I measure seven attributes, namely:

distance of King from corner
number of empty squares from King to closest corner
mobility
number of defenders on board
number of attackers on board
number of captured defenders
number of captured attackers

NOTE: the number of captured attackers and defenders measured is not the *total* captures, but rather the captures that resulted from that particular move.

Looking at the number of empty squares from the King to the closest corner was something I added to try and give the King more of an **incentive** when trying to move to a corner instead of worrying about pieces being captured.

Once all this data has been gathered, I then use weights to decide how much to emphasise certain attributes. They are as follows:

light	-500	-1000	+10	-1	+1	+200	-100
dark	+500	+1000	+10	+1	-1	-100	+200

These weights correspond to the ordering of the attributes measured shown earlier. The weights are multiplied by their respective attributes and added to the total value with the exception of the *"distance of King from corner"* and *"number of empty squares from the King to the closest corner."* Instead, these weights are **divided** by the attribute and added to the value. This is to reflect the fact that as the king's distance from the corner gets smaller, the value for the players should get larger either positively or *negatively* respectively.

I played around with these weights quite a bit and gave emphasis on *"distance of King from corner"* and *"number of empty squares from the King to the closest corner."* The weights are either positive or negative, depending on the difference to the player to move, and this is to return a value greater than 0 if good for the player, or less than 0 otherwise.

One possible solution to these weights was to play many games with random weights each time to see what works but I didn't have time to do this (although it would have been interesting to see the result!) Also, since we were asked to return an *integer* value, I decided not to multiply by fractions.

3 Alpha-Beta

As with the previous assignment, when alpha-beta nodes are printed, they are done so using the template:

---

alpha: . . . , beta: . . .
value: . . .
method: . . .

However, I also added notifications to the user when a

value in the **transposition-table** with a depth >= the current depth was used, or when normal Alpha-Beta pruning takes place. It prints in the same format as requested for the last assignment, and there is also on option at the beginning of the program to add the transposition-table.

Initial Move Choice When choosing a move, I simply find all available pieces for that player, and then all possible moves for those pieces and then amalgamate all the possible moves and start alpha-beta on each of them. Whichever move returned the highest result is chosen. Also, the **total number of static evaluations performed** is the addition of all static evaluations returned from alpha-beta on the amalgamation of all possible moves.

4 Zobrist Hashing and Transposition-Table

Array Initialization When initialising my array of random numbers, I decided to use **16bit** as 64 bits would not really be needed in this experiment as I don't need to check for collisions (i.e. if stored position is the same as a newcomer). Also, once generated, I then XOR all the numbers in the array to get the initial Zobrist Key, **but only once**.

Calculating New Key When calculating a new key, it is done as follows:

newZobristKey = ZobristKey
$\wedge$ **squareMovedFrom**
$\wedge$ **squareMovedTo**
$\wedge$ **anyCaptures**
$\wedge$ **fixedAmountForChangeOfPlayer**.

The fixedAmountForChangeOfPlayer is a random number I chose (i.e. 3) to reflect whose turn it is

Storage in Transposition Table When adding values to the Transposition-Table, I stored three values:

Key
Depth
Value

Also, extra space is created automatically in the transposition so didn't need to initialise space for 64K entries as it's already capable of holding that amount.

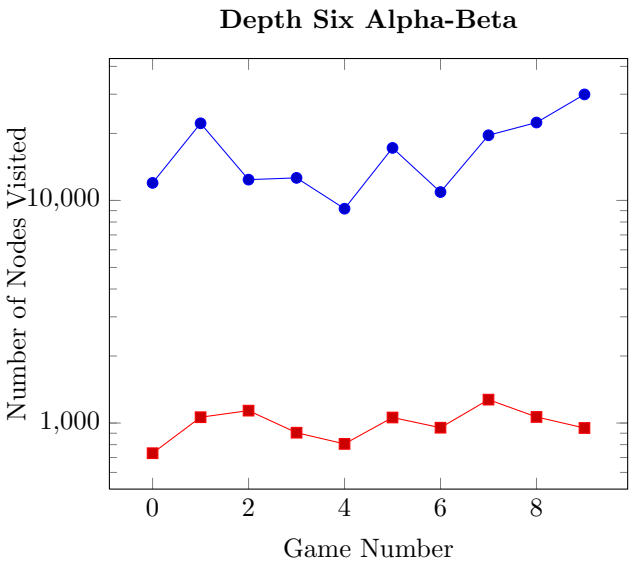
Lookup of Transposition Table When performing a key lookup, if the key wasn't present I simply create the entry and add the depth I was at, the number of static evaluations performed and the value returned. However, if the value was present, I look to see what the **Depth** value stored with that key is. If the depth is >= to my current depth, I use that value and don't call Alpha-Beta again but if it is less, I then run Alpha-Beta and replace all the values returned that were previously stored with that key with the values from that Alpha-Beta search.

5 Experiments

5.1 Depth 6 Alpha-Beta ($\alpha\beta$)

game #	#nodes visited (**no tt**)	#nodes visited (**tt**)
1	11899	732
2	22210	1062
3	12401	1137
4	12629	905
5	9181	806
6	17219	1059
7	10904	953
8	19627	1274
9	22386	1065
10	29946	950

Note: tt = Transposition-Table. Nodes Visited is an Average

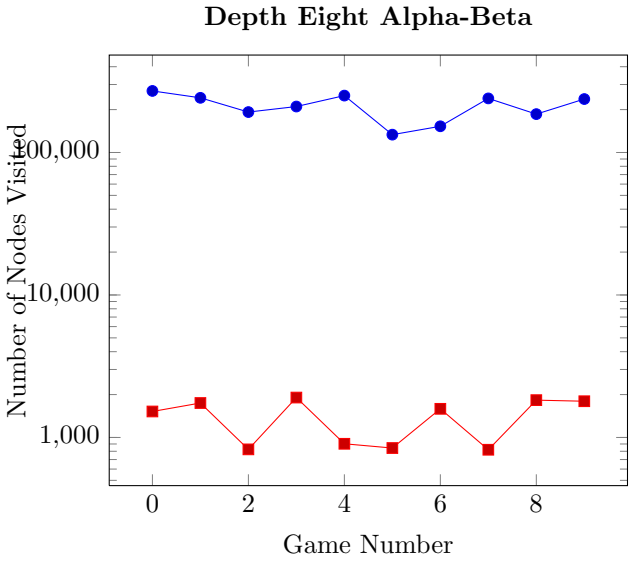


Note: red = transposition table, blue = pure alpha-beta

5.2 Depth 8 Alpha-Beta ($\alpha\beta$)

game #	#nodes visited (**no tt**)	#nodes visited (**tt**)
1	270412	1522
2	241929	1745
3	192221	825
4	210177	1906
5	250780	903
6	133400	843
7	152736	1589
8	239657	820
9	185940	1828
10	237209	1796

Note: tt = Transposition-Table. Nodes Visited is an Average



Note: red = transposition table, blue = pure alpha-beta

6 Conclusions

Following the experiments, it is clearly seen that the number of nodes which were visited during Alpha-Beta with the transposition-table enhancement were **significantly** less than using pure Alpha-Beta (NegaMax).

The moves for the computer were also averaging a great deal more seconds without the transposition-table but if used, the computer was able to return a move within about a second or two. This is obviously due to the key factor namely that there are far fewer nodes to search and also that entire sub-trees can sometimes be avoided altogether, greatly reduces the amount of time spent searching. With fewer nodes being searched, this also means that I didn't have to create as many board states, and so it was also much more effective on memory consumption. Searching to a very deep depth without a transposition-table will use massive amounts of memory, and so using one is a lot more efficient in terms of memory, and speed.

Sometimes the move made with the transposition table was indeed different than pure alpha-beta. One of the reasons for these different moves is possibly incorrect results being returned with the transposition-table due to the *Graph History Interaction Problem*. This is essentially where the same board state (game position) gives a different value when reached via different paths, meaning how you got to that board state affected to result.

An important lesson I learned is that it is a non-trivial exercise coming up with an effective evaluation function. Sometimes the computer was too tentative, other times too aggressive, and one way I tried ended up with the king simply moving around the board refusing capture. I tried to choose balancing weights after much game play, that give a noticeable difference between how good that board position is in respect to the player at hand.