CS76 21F - PA-3 Chess [Benjamin Cape]

Description

How do your implemented algorithms work? What design decisions did you make?

My implemented algorithms work exactly as the psuedocode is provided in the textbook/from the slides.

I implemented alpha beta pruning minimax with the same game structure as a normal minimax. So, any game that you write that can be passed into an alpha-beta pruning search, can also be used in a regular minimax search. I found this particularly helpful when testing the speed of alpha-beta with respect to normal minimax.

The iterative deepening algorithm is also interesting, in that it takes a time value as a toggle, and deepens as long as the previous search took less than the required time.

The only other algorithm that I introduced which is of substance is the evaluation function. Which I will talk about in the discussion question.

Evaluation

Do your implemented algorithms actually work? How well? If it doesn't work, can you tell why not? What partial successes did you have that deserve partial credit?

Yes, my algorithms for search work very well. The alpha beta pruning performs very well as compared to the regular minimax, allowing us to search much deeper into the search tree.

Discussion

1. (minimax and cutoff test) Vary maximum depth to get a feeling of the speed of the algorithm. Also, have the program print the number of calls it made to minimax as well as the maximum depth. Record your observations in your document.

```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
---------------
```

```
a b c d e f g h

White to move

Minimax visited 422 nodes, with depth: 2, in 11.893ms
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . N
P P P P P P P P
R N B Q K B . R
---------------
a b c d e f g h

Black to move

Minimax visited 207805 nodes, with depth: 4, in 526.411ms
```

It is clear that the deeper the depth, the more nodes we visit. Also, clearly observable by the time here, it takes MUCH longer if our depth is higher. This is because our branching factor is VERY high, so each added depth increases our search results exponentially.

2. (evaluation function) Describe the evaluation function used and vary the allowed depth, and discuss in your document the results.

My evaluation function is simple. It assigns a score to each piece:

```
scores = {
    chess.PAWN: 1,
    chess.ROOK: 5,
    chess.KNIGHT: 3,
    chess.BISHOP: 3,
    chess.QUEEN: 9,
}
```

There also is a maximum score: 8 + 2 * (5 + 3 + 3) + 9 = 39

Then, a linear sum of the frequency of each root color, and the score for the piece, minus the same value for the not root color pieces followed by normalization over the maximum possible score provides us a number $[-1, 1]$ that indicates a score for a particular individual.

If the root wins, they get a score of 1, if they lose, they get a score of -1.

From the start state, even with a deep depth-limit (5):

```
r n b q k b n r
```

```
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
---------------
a b c d e f g h
```

```
Minimax visited 5072214 nodes, with depth: 5, in 893.37ms, best move value: 1.0
```

We know this makes sense, because there IS a way to keep all of your pieces with 5 moves in a game of chess.

3. (alpha-beta) Record your observations on move-reordering in your document.

With Ordering:

```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
---------------
a b c d e f g h

White to move

Alpha Beta Pruning Visited: 13059 nodes, with depth: 5, in 950.936ms, best move value: 1.0
```

Without Ordering:

```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R
---------------
a b c d e f g h
```

```
White to move
```

```
Alpha Beta Pruning Visited: 13181 nodes, with depth: 5, in 360.546ms, best move value: 1.0
```

The ordering saves us some nodes, a.k.a it allows us to prune more I guess. But it takes much longer because we have to sort the list of actions and their expected values. This probably makes more sense to do later in the game though, not at the beginning.

4. (iterative deepening) Verify that for some start states, best_move changes (and hopefully improves) as deeper levels are searched. Discuss the observations in your document.

For the chess starting state, an alpha-beta pruner can get the following 2 different result:

```
Alpha Beta Pruning Visited: 17528 nodes, with depth: 5, in 510.486ms, best move value: 0.025
----
Alpha Beta Pruning Visited: 1516 nodes, with depth: 4, in 56.544ms, best move value: 0.0
```

By searching one depth further, we get a slightly better outcome. Strange though, if we search one depth further, to 6, we get a negative payout. ** This might mean something is wrong with how I'm calculating the score **

Extra Credit

For extra credit I have implemented the rock-paper-scissors checkers game created by Professor Bjoern. In order to accomplish this, I have implemented multi-agent maximization as well.

I have also implemented a transposition table.