Author: Zitong Wu

Course: COSC 276 Artificial Intelligence

Term: Fall 2020

Report – PA3 Chess

Description

1. The minimax algorithm is a recurisve depth first search algorithm. It works by calling the `max_value` and `min_value` functions in turn. The `max_value` function represents the player that tries to maximizes its score and the `min_value` function represents the player that tries to minimize the score of its opponent. The utility function is called when the search reaches the terminal. Alpha-beta pruning helps cut unpromising subtrees. The alpha value keeps track of the best value found so far for the max player, and the beta value keeps track of the best value found so far for the min player (from the perspective of the min player). In alpha-beta minimax search, we replace the terminal test with the cutoff test to reduce the search depth, and replace the utility function with the evaluation function that returns an estimate of the true utility value.
2. Other than following the algorithm, the most critical design decision is random shuffling the legal moves at the beginning of the minimax search. Without doing this, the minimaxAI will choose repeated moves and the game will just go on and on and end in a draw.

Evaluation

1. My implemented algorithm works. For example, an alphabetaAI of depth 3 always beats an alphabetaAI of depth 1 in 50 moves on average. For another example, an alphabetaAI of depth 3 always beats a randomAI in 50 moves on average as well (based on the test results from `test_average_performace.py`).
2. One thing that didn't work as expected was sorting the moves to be explored at each node in alpha-beta minimax search. It is supposed to help with pruning more nodes and thus cutting down the run time. However, in my case, it is not faster than just using random shuffling once at the beginning of the search, which is the version I ended up adopting. This is because the benefit of doing the sorting is cancelled out by the time it takes to do the soriting. Sorting would help if the power of evaluation function outweighs the cost of sorting.

Discussion Questions

Minimax and Cutoff Test

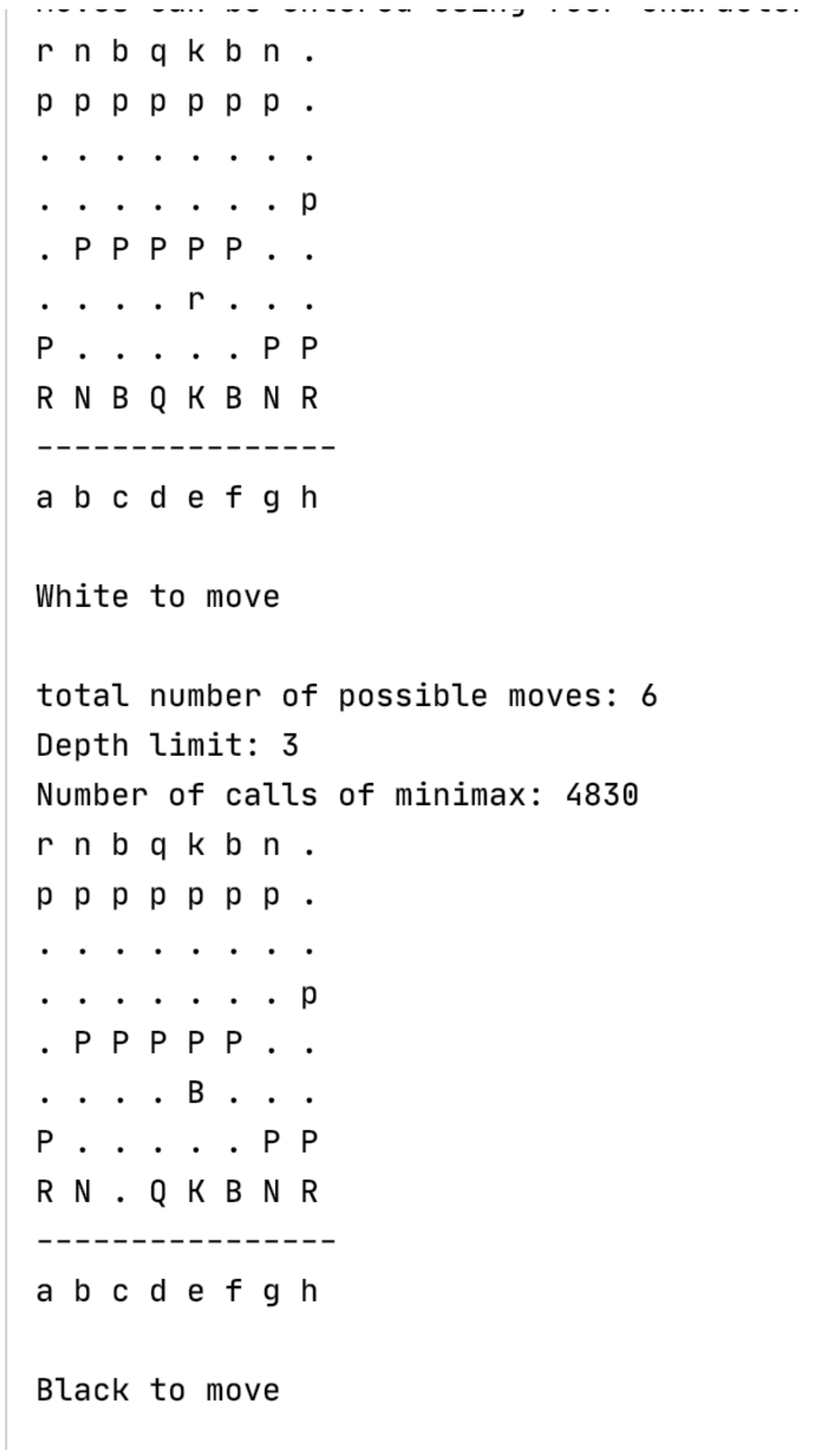
1. At depth 1, a minimaxAI takes around 0.005 second to find the best move and makes 25-30 calls to the minimax function.
   At depth 2, a minimaxAI takes around 0.1 second to find the best move and makes 250-1000 calls to the minimax function.
   At depth 3, a minimaxAI takes around 2 seconds to find the best move and makes 10000-350000 calls to the minimax function.
   At depth 4, a minimaxAI takes around 30 seconds to find the best move and makes more than 200000 calls to the minimax function. This is already taking too long.

Evaluation Function

1. My evaluation functions consists of two parts. The first part sums the material values of the pieces left for the minimaxAI minus the material values of the pieces left for the opponent. The second part adds a high value (100) to the evaluation value if the minimaxAI checkmates the opponent, or it substracts a high value(100) from the evaluation function if the minimaxAI is checkmated by the opponent. It is important to note that I use `board.is_checkmate()` rather than `board.is_check()` for the second part of my evaluation function. When I uses board.is_check() and have an alphabetaAI of depth 3 play against an alphabetaAI of depth 1, it always comes to the situation where towards the end of a game there are many pieces left for the alphabetaAI of depth 3 and they are chasing around the only king left for the alphabetaAI of depth 1. This is because checking the opponent is not a move that is good enough. One side can always check the other side, and the other side can always evade the capture, and this can go on for a long time, making the total number of moves higher than using `board.is_checkmate`() in the evaluation function. I wrote a simple program in `test_average_perfomance.py` to test 20 trials of the game staging an alphabetaAI of depth 3 against an alphabetaAI of depth 1. When I use `board.is_check()` in my evaluation function, the total number of moves for a game is about 200 on average, whereas when I use `board.is_checkmate()` in my evaluation function, the total number of moves for a game is only about 50 on average.

2. With the evaluation function, the minimaxAi is now hard to beat. I created the following board position and had a minimaxAI of depth 3 (white) play against a minimaxAI of depth 1 (black). In this board position (fen string = rnbqkbn1/ppppppp1/8/7p/1PPPPP2/4r3/P5PP/RNBQKBNR),

the black is checking the white. But the white successfully blocks this check by capturing the black rook with the white bishop.



```
Moves can be entered using four character
r n b q k b n .
p p p p p p p .
. . . . . . . .
. . . . . . . p
. P P P P P . .
. . . . r . . .
P . . . . . P P
R N B Q K B N R
----------------
a b c d e f g h

White to move

total number of possible moves: 6
Depth limit: 3
Number of calls of minimax: 4830
r n b q k b n .
p p p p p p p .
. . . . . . . .
. . . . . . . p
. P P P P P . .
. . . . B . . .
P . . . . . P P
R N . Q K B N R
----------------
a b c d e f g h

Black to move
```

The following shows the end of this game. The white correctly chooses the move that checkmates the black king with the white queen.

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

White to move

total number of possible moves: 39
Depth limit: 3
Number of calls of minimax: 14976
k . Q . . . . .
p . . . . . B .
. . . . . . . .
. p . . . P . .
. . p . . P . .
. . . . . . . P
P . . . . . . P
R N . . K . N R
----------------
a b c d e f g h

Black to move

White wins
```

3. A minimaxAI takes longer time to run, so I use alphabetaAI instead to test different depth limits. As mentioned above, I wrote a simple program in `test_average_perfomance.py` that can test 20 trials of a game and output the average number of moves for the game. Here's the result:

- AlphabetaAI of depth 2 vs. alphabetaAI of depth 1: the total number of moves for a game is about 65.

- AlphabetaAI of depth 3 vs. alphabetaAI of depth 1: the total number of moves for a game is about 50.

- AlphabetaAI of depth 4 vs. alphabetaAI of depth 1: the total number of moves for a game is also about 50.

- As we can see, having the same opponent alphabetaAI of depth 1, an alphabetaAI of depth 3 ends a game faster than an alphabetaAI of depth 2. However, an alphabetaAI of depth 4 does not end a game faster than the alphabetaAI of depth 3. They have about the same performance according to my test. I think it may be because my evaluation function is still very elementary. When the evalutive power is limited, no matter how much I increase the search depth, the overall performance will be limited.

Alpha-Beta Pruning

1. Yes, with alpha-beta pruning, it takes less time to search for a move. It can now search to depth 5.

- An alphabetaAI of depth 3 now takes only 0.5 second to find the best move, while a minimaxAI of depth 3 takes 2 seconds to find the best move.
- An alphabetaAI of depth 4 now takes only 1-10 seconds to find the best move, while a minimaxAI of depth 4 takes 30 seconds to find the best move.
- An alphabetaAI of depth 5 takes 10-30 seconds to find the best move, while a minimaxAI of depth 5 takes much longer time to find the best move (I didn't wait until it found the move).
- Depth 5 is about the limit of an alphabetaAI with random shuffling the list of legal moves at the beginning of the search.

2. Sorting the moves to be explored at each node by their evaluation values does not really help searching faster in my case, because sorting takes time and my evaluation function is quite elementary. The benefit of doing sorting is cancelled out by the cost of doing the sorting, unless the evaluation function is powerful. Now an alphabetaAI of depth 4 takes 1-5 seconds to find the best move, compared to 1-10 seconds without sorting. An alphabetaAI of depth 5 takes 10-60 seconds, compared compared to 10-30 seconds without sorting.

3. Given the same initial board position and the same depth limit, my minimaxAI and alphabetaAI each returns a move with the same value, and my alphabetaAI always explores equal or fewer number of nodes (see the "Number of calls of minimax" in the printout) than the minimaxAI. For the following board position (fen string: r1bqk3/pppp4/2n1Pp2/PPP1P2p/5pn1/2N1r3/3b2PP/R1BQKBNR), I tested minimaxAI and alphabetaAI of depth 1, 2, and 3.

- Both minimaxAI and alphabetaAI of depth 1 give a move with value 3. Both explore two nodes (2 calls to the minimax function).

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

White to move

Total number of moves: 0
MinimaxAi of depth 1
time used: 0.00024104118347167969
total number of moves: 1
best value: 3
best move: e1d2
Number of calls of minimax: 2
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
----------------
a b c d e f g h
```

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

White to move

Total number of moves: 0
AlphabetaAI of depth 1
time used: 0.00023698806762695312
best value: 3
best move: e1d2
Number of calls of minimax: 2
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
---------------
a b c d e f g h
```

- Both minimaxAI and alphabetaAI of depth 2 give a move with value 0. Both explore 34 nodes (34 calls to the minimax function).

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

White to move

Total number of moves: 0
MinimaxAi of depth 2
time used: 0.004380226135253906
total number of moves: 1
best value: 0
best move: e1d2
Number of calls of minimax: 34
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
---------------
a b c d e f g h
```

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

White to move

Total number of moves: 0
AlphabetaAI of depth 2
time used: 0.004612922668457031
best value: 0
best move: e1d2
Number of calls of minimax: 34
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
----------------
a b c d e f g h
```

- Both minimaxAI and alphabetaAI of depth 3 gives a move with value 2. The minimaxAI explores 1067 nodes (1067 calls to the minimax function), whereas the alphabetaAI only explores 388 nodes (388 calls to the minimax function).

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

White to move

Total number of moves: 0
MinimaxAi of depth 3
time used: 0.1481170654296875
total number of moves: 1
best value: 2
best move: e1d2
Number of calls of minimax: 1067
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
----------------
a b c d e f g h
```

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

White to move

Total number of moves: 0
AlphabetaAI of depth 3
time used: 0.05437898635864258
best value: 2
best move: e1d2
Number of calls of minimax: 388
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
---------------
a b c d e f g h

Black to move
```

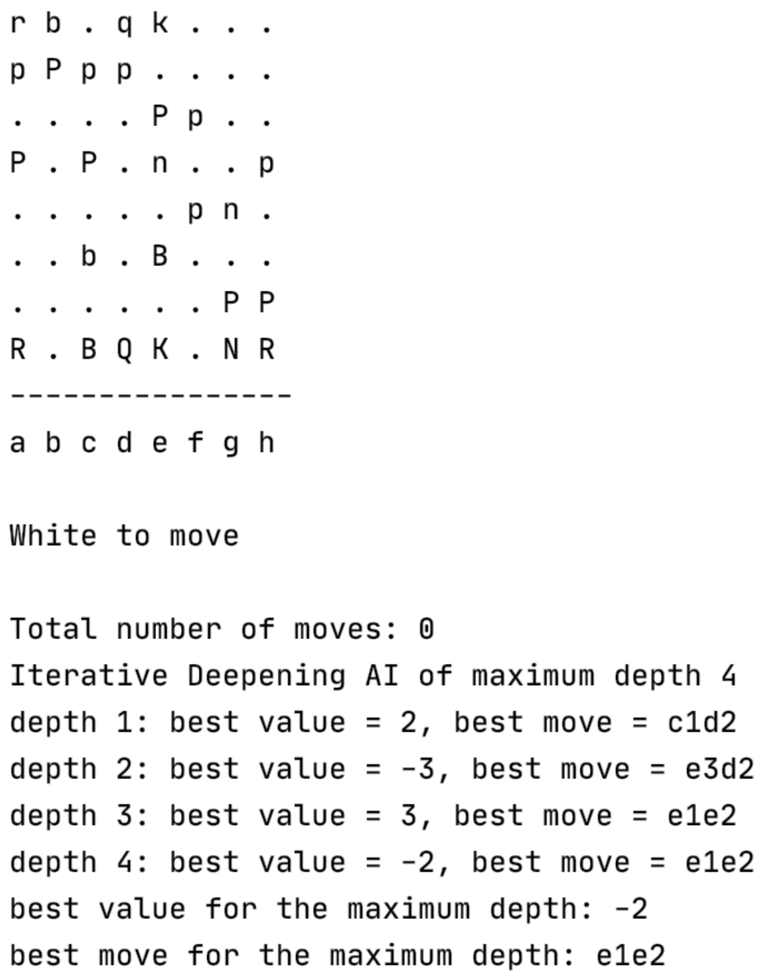
- Both minimaxAI and alphabetaAI of depth 4 gives a move with value -1. The minimaxAI explores 33254 nodes (33254 calls to the minimax function), whereas the alphabetaAI only explores 4612 nodes (4612 calls to the minimax function).

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

White to move

Total number of moves: 0
MinimaxAi of depth 3
time used: 0.1481170654296875
total number of moves: 1
best value: 2
best move: e1d2
Number of calls of minimax: 1067
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
----------------
a b c d e f g h
```

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

White to move

Total number of moves: 0
AlphabetaAI of depth 3
time used: 0.05437898635864258
best value: 2
best move: e1d2
Number of calls of minimax: 388
r . b q k . . .
p p p p . . . .
. . n . P p . .
P P P . P . . p
. . . . . p n .
. . N . r . . .
. . . K . . P P
R . B Q . B N R
---------------
a b c d e f g h

Black to move
```

Iterative Deepening

For the following board position (fen number: rb1qk3/pPpp4/4Pp2/P1P1n2p/5pn1/2b1B3/6PP/R1BQK1NR) the iterative deepening search at different depth gives different best move suggestions and different evaluation values. I cannot comment on whether the move suggested by the deepest search is the best, because I don't know much about chess.



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

White to move

Total number of moves: 0
Iterative Deepening AI of maximum depth 4
depth 1: best value = 2, best move = c1d2
depth 2: best value = -3, best move = e3d2
depth 3: best value = 3, best move = e1e2
depth 4: best value = -2, best move = e1e2
best value for the maximum depth: -2
best move for the maximum depth: e1e2
```