Algorithm: Limited-depth Alpha-beta Pruning (for chess)

Alpha-beta pruning is a recursive algorithm for finding an optimal move as defined by a deterministic heuristic score (up to a maximum search depth) in zero-sum 2-player games. It improves upon the performance of minimax by keeping track of the lower bounds on each player's potential set of moves and pruning the decision tree when a better move exists.

Time Complexity:

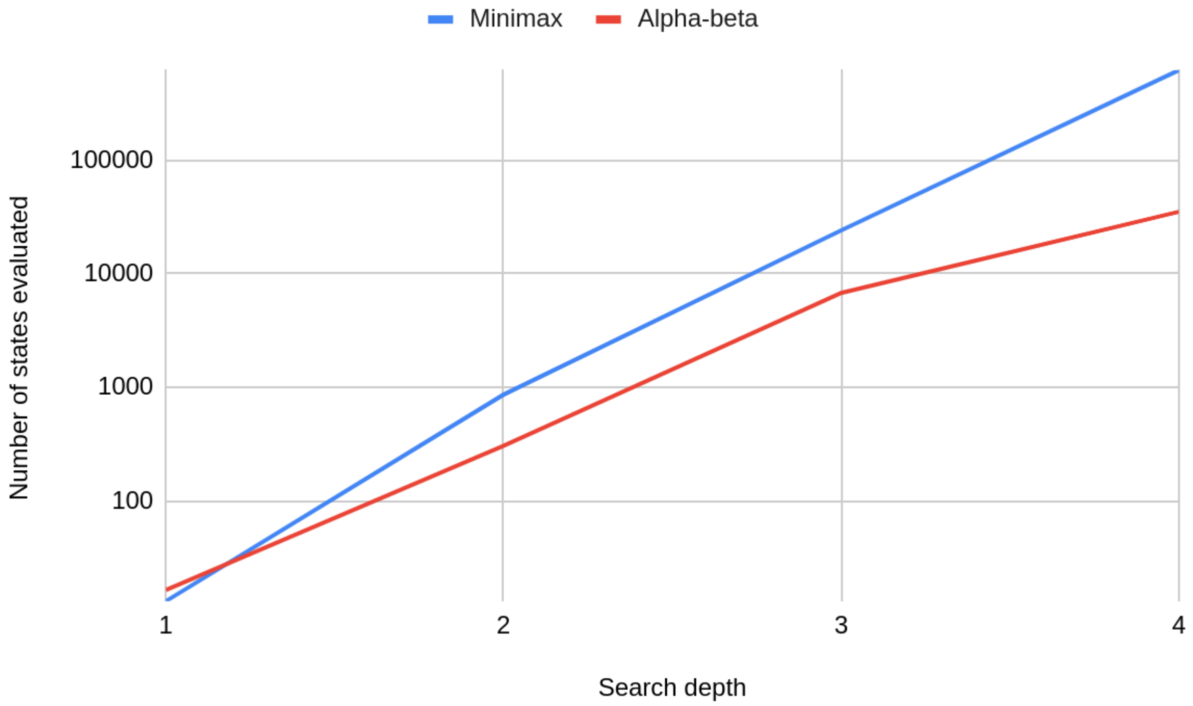
Best case: $O(sqrt(b^d)) = O(b^{\frac{d}{2}})$

Worst case: $O(b^d)$

where b is the branching factor, and d is the search depth.

The average runtime is in between the best and worst case with $O(b^{kd})$ where k is between ½ and 1.

Empirical Runtime Analysis:



To analyze the performance of alpha-beta pruning, we can examine the average number of states visited as a function of the maximum search depth, in comparison to the minimax algorithm. The following data is based on a sample of 10 games for each search depth from the starting position:

	Average number of states evaluated per move				
Maximum search depth (d)	Minimax: O(b^d)	Alpha-beta: O(b^{kd})	Ratio	Estimated branching factor (b)	Estimated Pruning factor (k)
1	13	14	0.9	13	1.0
2	862	305	2.8	29	0.85
3	24,087	6,774	3.6	29	0.87
4	619,777	35,069	17.7	28	0.79

Let N be the number of states evaluated by alpha-beta pruning. Since $N \approx b^{kd}$, where $\frac{1}{2} \le k \le 1$, we can solve for the pruning factor:

$$k \approx \log_b(N)/d = \log(n)/(d\log(b))$$

The ratio of minimax and alpha-beta’s number of states evaluated is:

$$b^d/b^{kd} = b^{d-kd} = b^{d(1-k)}$$

For depth d = 4, we found k = 0.79, which is better than the unpruned minimax, but not quite optimal pruning as k is still greater than 0.5.

File Directory Structure:

The input files are located in the *test_data* subdirectory.
The expected output files are located in the *expected_output* subdirectory, to be validated against *moves_played.txt* (which is generated in the project’s base directory by running *linux_tester.c*).

Below is a description of each source file:

Source file name	Description
ai.c	4 AIs are implemented: random (0-ply), greedy (1-ply), minimax (n-ply), and alpha-beta (n-ply), for play and testing.
analysis.c	functions for evaluating the game state and value (heuristic score)
board.c	functions to initialize an empty board and place pieces, as well as helper functions for converting between coordinate and algebraic board notations
linux_io.c	contains the main function which controls the game flow and input/output to both terminal (user interaction) and files (*moves_played.txt*, *[AI]_output_[white|black]_player.txt*)
linux_tester.c	a bare-bones version of *linux_io.c* that accepts a command line argument for choosing the input test file, used for more convenient testing of the alpha-beta based AI in self-play. Test cases should be run using *linux_tester.c* rather than *linux_io.c.*
pieces.c	contains the game logic including which moves each piece can make in chess, while avoiding illegal moves.
player.c	defines player types (black or white) and switches turns between them

util.c	defines the board coordinates and piece labels
windows_game.c	terminal-based UI for playing the game with human or AI.
windows_ui.c	basic UI library for displaying the chess board in terminal

All code was implemented in C (except the Makefile).

File Input Specification:

- The first line of input will have a positive integer for Alpha-beta's maximum search depth (i.e. number of plies).
- The second line of input will have a positive integer N for the number of moves to play.
- The next 8 rows will each contain 8 characters that are either a space (' ') for an empty square, or a piece type.
    - Each piece type is represented by a single character:

Piece	White	Black
Knight	N	n
Queen	Q	q
King	K	k
Bishop	B	b
Rook	R	r
Pawn	P	p

    - White's pieces are in uppercase and black's pieces are in lowercase.
    - This is the starting position from which the AI will play the game against itself.
- These test cases can be visualized by the Lichess board editor:
    - https://lichess.org/editor/4b2k/6pp/5Q2/8/8/8/8/7K_w_-_-_0_1 (*test_data/input0.txt*)
    - https://lichess.org/editor/Qn5k/8/2r5/3b4/8/8/PP6/KB6_w_-_-_0_1 (*test_data/input1.txt*)

- https://lichess.org/editor/k7/pp6/1p6/4qr1p/6B1/6P1/6PP/6BK_w_-_-_0_1 (*test_data/input2.txt*)
- https://lichess.org/editor/r3q2k/pb1p1p1B/1p1b1p1B/2pP1Q2/2P2P2/8/PP4PP/6K1_w_-_-_0_23 (*test_data/input3.txt*)

File Output Specification:

- The output will contain N lines. Each line contains a string in the format `ab-cd`, where `ab` is the square of the piece that moved and `cd` is its destination after the move, in algebraic notation. The characters `a` and `c` are letters between ‘a’ and ’h’, inclusive, while `b` and `d` are digits between 1 and 8, inclusive. (Note that captures are referred to in the same way.)
- To verify the correctness of outputs, I compared alpha-beta’s chosen move against minimax’s as well as Stockfish’s official solution on Lichess.
- The expected outputs from *moves_played.txt* are stored in the subdirectory named *expected_output*.

How to Compile and Run:

To compile and build with gcc:

```
$ make all
```

To run a test on Linux:

```
$ ./linux_tester test_data/[input_file.txt]
```

where *input_file.txt* is the test you want to run.

To play the game for fun on Linux (test I/O only, no UI):

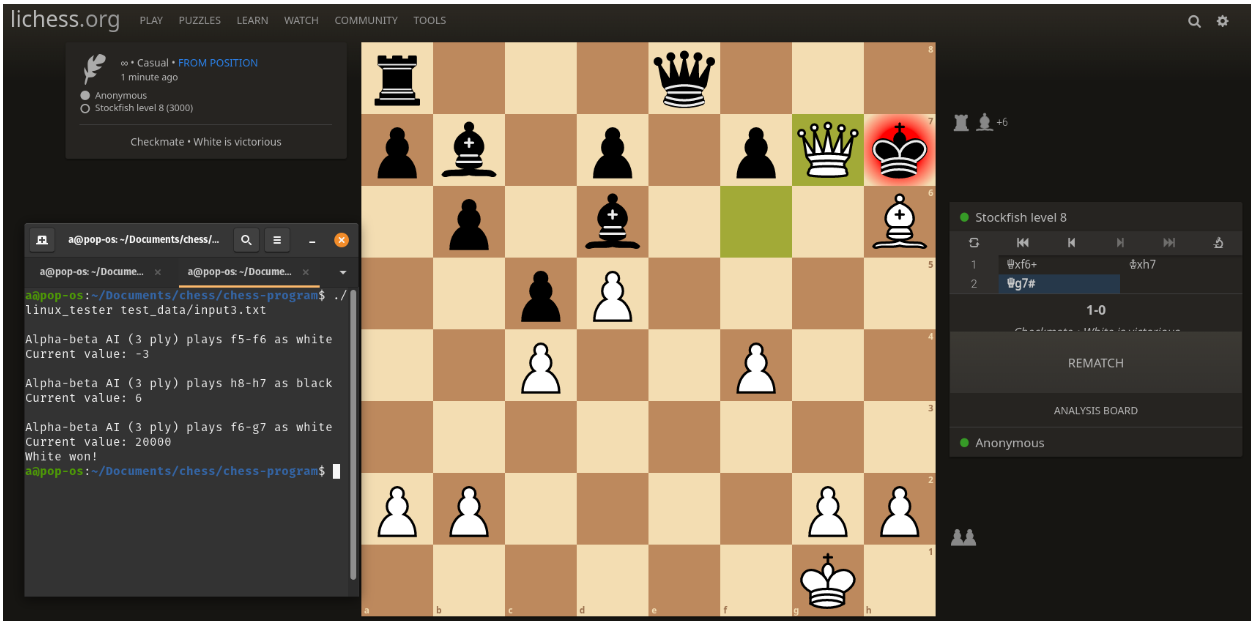
```
$ ./linux_io
```

To play the game seriously on Windows with basic terminal UI:

```
$ windows_game
```

Appendix: Screenshots

(Running the *linux_tester* and validating the output against Lichess.org’s Stockfish engine.)



∞ • Casual • FROM POSITION
1 minute ago
● Anonymous
○ Stockfish level 8 (3000)

Checkmate • White is victorious

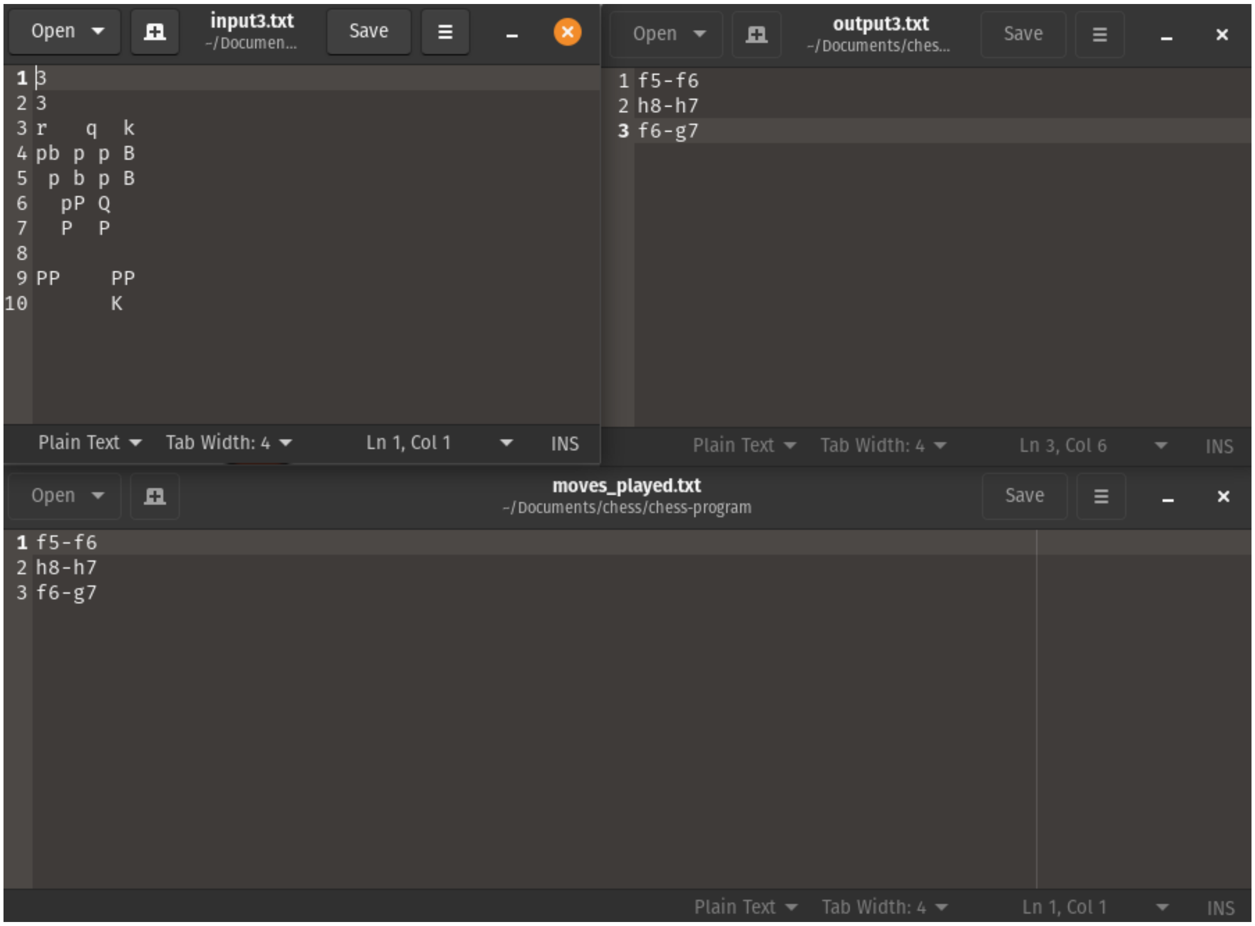
```
a@pop-os:~/Documents/chess/chess-program$ ./
linux_tester test_data/input3.txt

Alpha-beta AI (3 ply) plays f5-f6 as white
Current value: -3

Alpha-beta AI (3 ply) plays h8-h7 as black
Current value: 6

Alpha-beta AI (3 ply) plays f6-g7 as white
Current value: 20000
White won!
a@pop-os:~/Documents/chess/chess-program$ 
```

● Stockfish level 8

```
1   ♕xf6+   ♚xh7
2   ♕g7#
```

1-0

● Anonymous



input3.txt

```
3
3
r   q  k
pb p p B
 p b p B
  pP Q
  P  P

PP    PP
      K
```

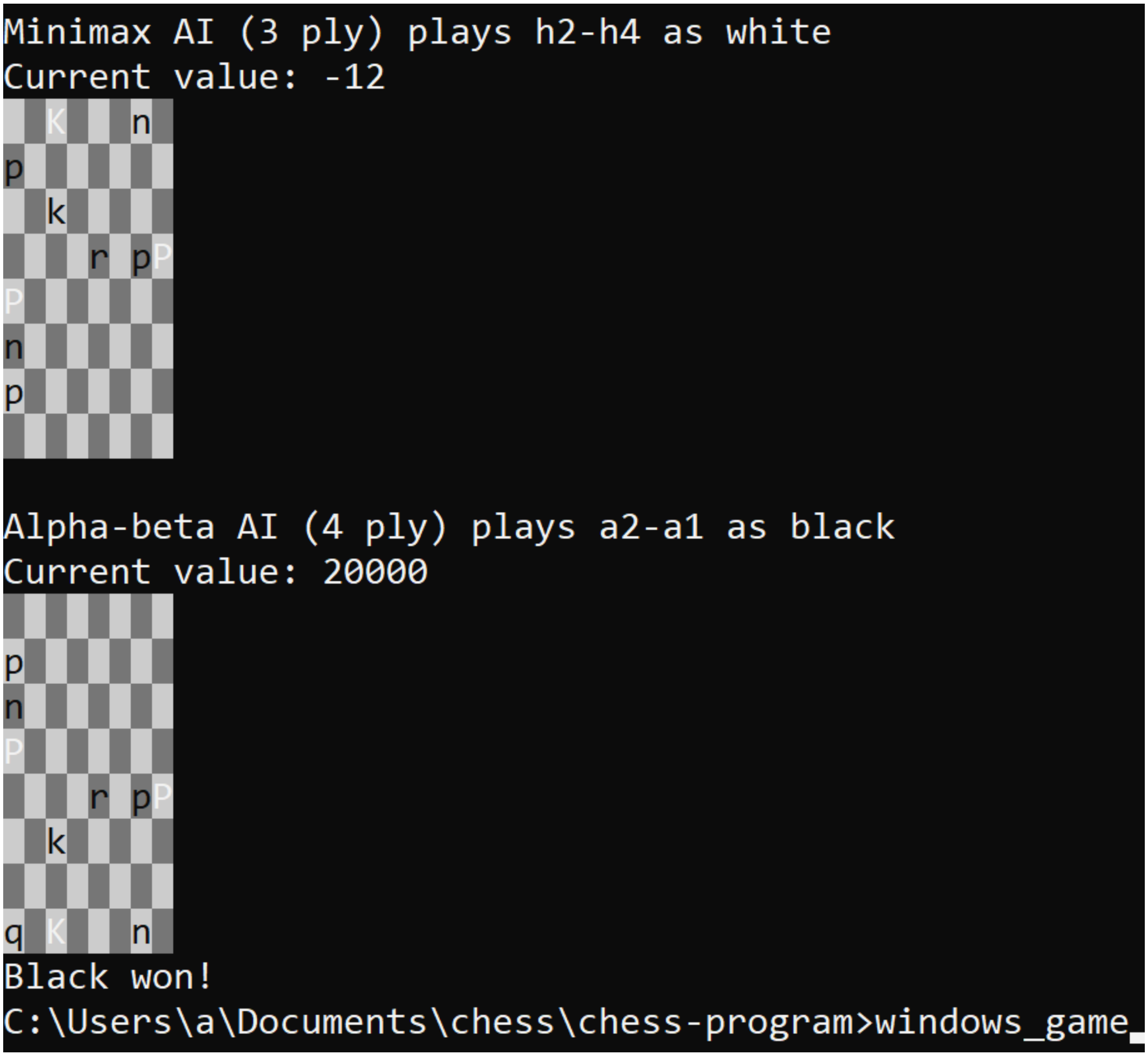
output3.txt

```
f5-f6
h8-h7
f6-g7
```

moves_played.txt

```
f5-f6
h8-h7
f6-g7
```

(Playing the game on Windows with Minimax 3-ply vs Alpha-beta 4-ply)



```
Minimax AI (3 ply) plays h2-h4 as white
Current value: -12
  K   n 
p       
  k     
    r pP
P       
n       
p       
        

Alpha-beta AI (4 ply) plays a2-a1 as black
Current value: 20000
        
p       
n       
P       
    r pP
  k     
        
q K   n 
Black won!
C:\Users\a\Documents\chess\chess-program>windows_game
```