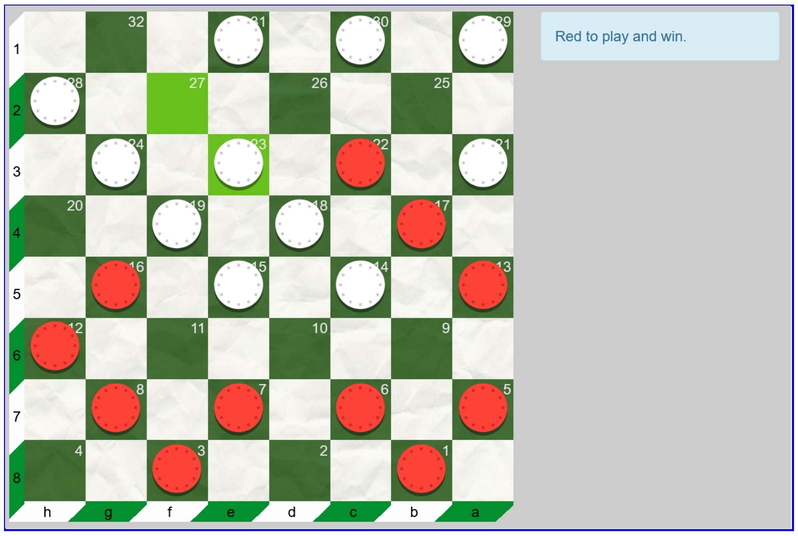
6. Compare the effect of increasing search depth (come up with a method to demonstrate your point).

Unsurprisingly, the more you increase the search depth, the better the AI becomes. Some states that were previously resulted in losses or ties become wins. Furthermore, the search time increases exponentially every time search depth is incremented.

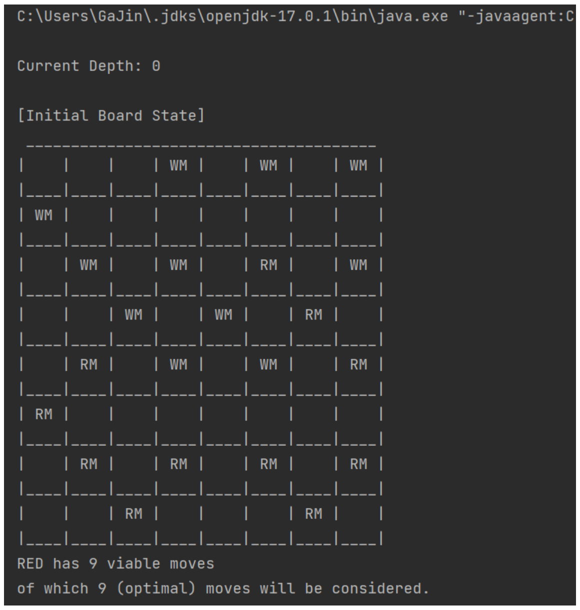
I searched online for checkers puzzles, and found a solvable state that's guaranteed to result in a victory for player red (given enough search depth).



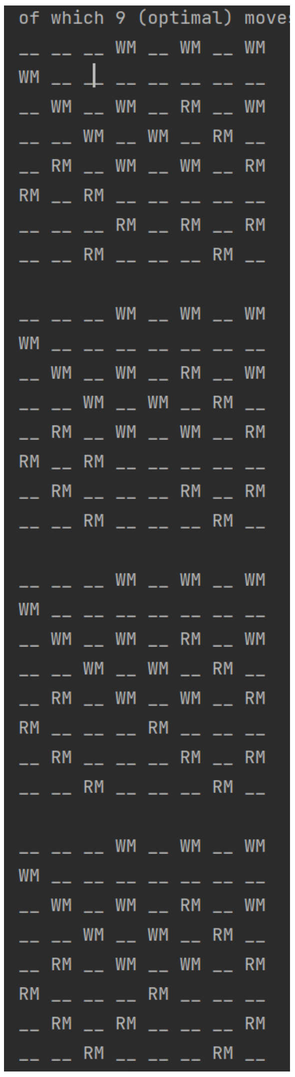
[link to puzzle]

I incremented my AI's search depth to see how it would gradually narrow down the number of "optimal moves" given its depth & basic heuristic.

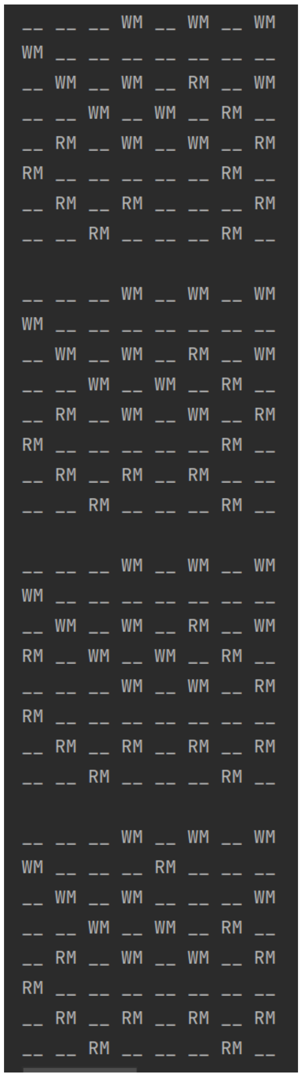
At a depth of 0, the AI just picks a random legal move



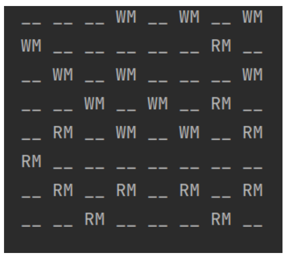
```
C:\Users\GaJin\.jdks\openjdk-17.0.1\bin\java.exe "-javaagent:C

Current Depth: 0

[Initial Board State]
 _______________________________________
|    |    |    | WM |    | WM |    | WM |
|____|____|____|____|____|____|____|____|
| WM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | WM |    | WM |    | RM |    | WM |
|____|____|____|____|____|____|____|____|
|    |    | WM |    | WM |    | RM |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | WM |    | WM |    | RM |
|____|____|____|____|____|____|____|____|
| RM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | RM |    | RM |    | RM |
|____|____|____|____|____|____|____|____|
|    |    | RM |    |    |    | RM |    |
|____|____|____|____|____|____|____|____|
RED has 9 viable moves
of which 9 (optimal) moves will be considered.
```



```
of which 9 (optimal) moves
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ RM __ __ __ __ __
__ __ __ RM __ RM __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ RM __ __ __ __ __
__ RM __ __ __ RM __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ RM __ __ __
__ RM __ __ __ RM __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ RM __ __ __
__ RM __ RM __ __ __ RM
__ __ RM __ __ __ RM __
```

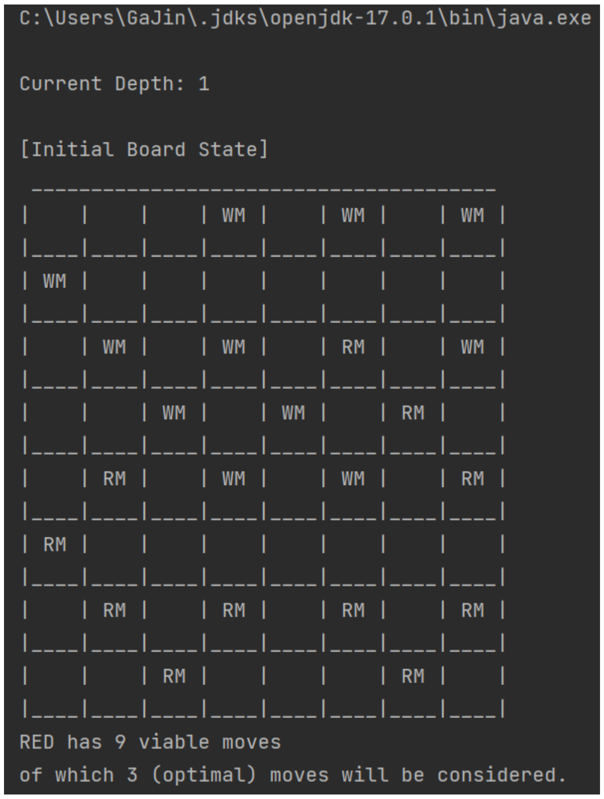


```
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ RM __
__ RM __ RM __ __ __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ RM __
__ RM __ RM __ RM __ __
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
RM __ WM __ WM __ RM __
__ __ __ WM __ WM __ RM
RM __ __ __ __ __ __ __
__ RM __ RM __ RM __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ RM __ __ __
__ WM __ WM __ __ __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ __ __
__ RM __ RM __ RM __ RM
__ __ RM __ __ __ RM __
```

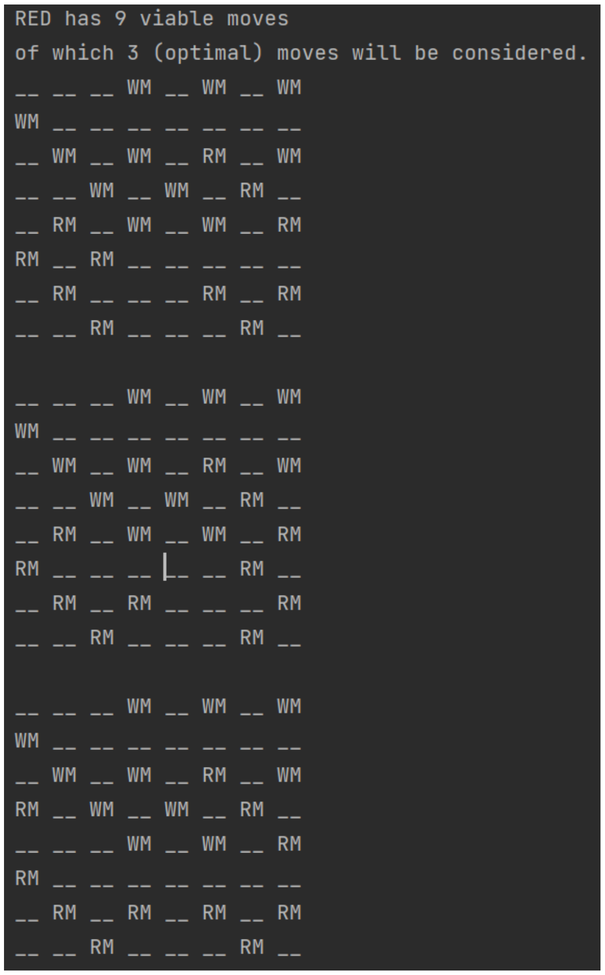


```
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ RM __
__ WM __ WM __ __ __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ __ __
__ RM __ RM __ RM __ RM
__ __ RM __ __ __ RM __
```

At a depth of 1, the AI no longer considers moves that immediately result in a piece loss (heuristic is based on the difference of red and white pieces on board)

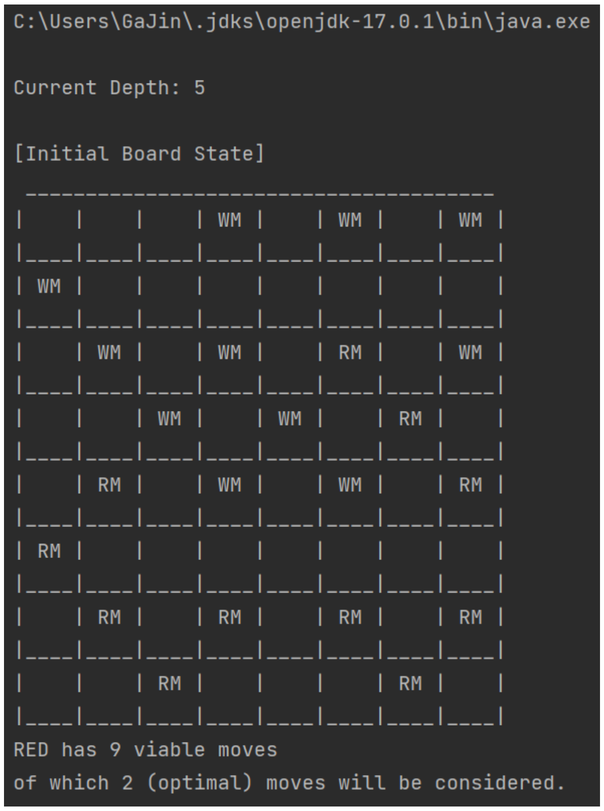


```
C:\Users\GaJin\.jdks\openjdk-17.0.1\bin\java.exe

Current Depth: 1

[Initial Board State]
 _______________________________________
|    |    |    | WM |    | WM |    | WM |
|____|____|____|____|____|____|____|____|
| WM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | WM |    | WM |    | RM |    | WM |
|____|____|____|____|____|____|____|____|
|    |    | WM |    | WM |    | RM |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | WM |    | WM |    | RM |
|____|____|____|____|____|____|____|____|
| RM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | RM |    | RM |    | RM |
|____|____|____|____|____|____|____|____|
|    |    | RM |    |    |    | RM |    |
|____|____|____|____|____|____|____|____|
RED has 9 viable moves
of which 3 (optimal) moves will be considered.
```

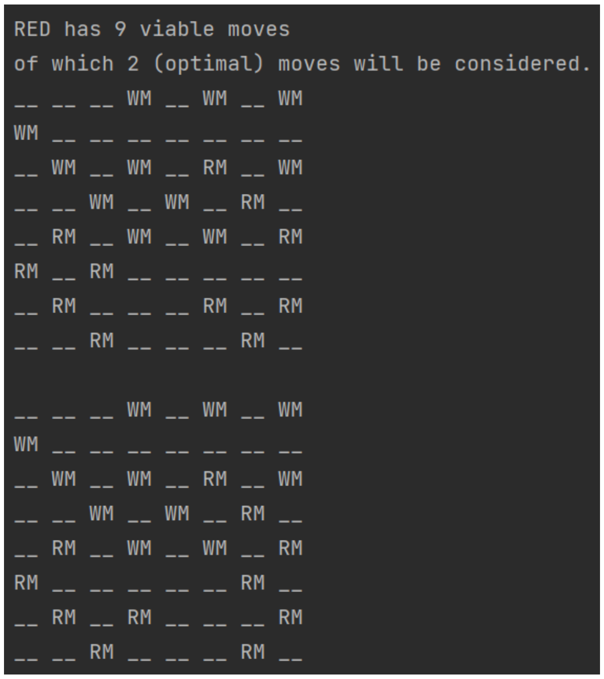


```
RED has 9 viable moves
of which 3 (optimal) moves will be considered.
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ RM __ __ __ __ __
__ RM __ __ __ RM __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ RM __
__ RM __ RM __ __ __ RM
__ __ RM __ __ __ RM __


__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
RM __ WM __ WM __ RM __
__ __ __ WM __ WM __ RM
RM __ __ __ __ __ __ __
__ RM __ RM __ RM __ RM
__ __ RM __ __ __ RM __
```

At a depth of 5, it narrows down the formerly three optimal moves to just two.

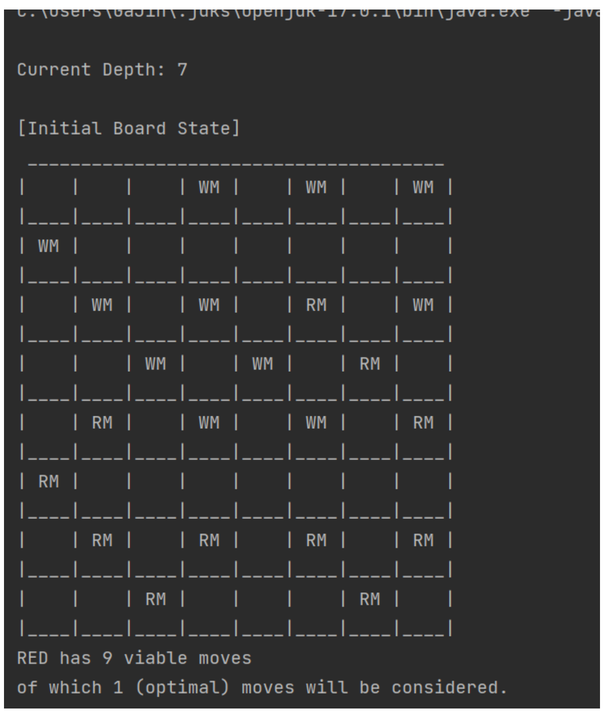


```
C:\Users\GaJin\.jdks\openjdk-17.0.1\bin\java.exe

Current Depth: 5

[Initial Board State]

 ---------------------------------------
|    |    |    | WM |    | WM |    | WM |
|____|____|____|____|____|____|____|____|
| WM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | WM |    | WM |    | RM |    | WM |
|____|____|____|____|____|____|____|____|
|    |    | WM |    | WM |    | RM |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | WM |    | WM |    | RM |
|____|____|____|____|____|____|____|____|
| RM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | RM |    | RM |    | RM |
|____|____|____|____|____|____|____|____|
|    |    | RM |    |    |    | RM |    |
|____|____|____|____|____|____|____|____|
RED has 9 viable moves
of which 2 (optimal) moves will be considered.
```

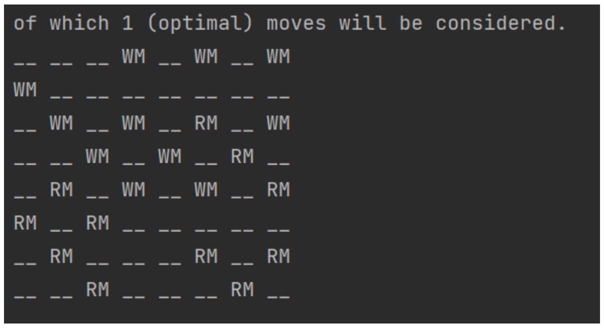


```
RED has 9 viable moves
of which 2 (optimal) moves will be considered.
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ RM __ __ __ __ __
__ RM __ __ __ RM __ RM
__ __ RM __ __ __ RM __

__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ __ __ __ __ RM __
__ RM __ RM __ __ __ RM
__ __ RM __ __ __ RM __
```

At a depth of 7, the AI only considered 1 move to be optimal



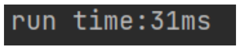
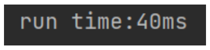
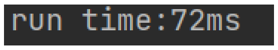
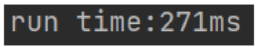
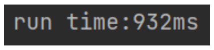
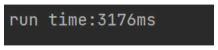
```
C:\Users\GaJin\.jdks\openjdk-17.0.1\bin\java.exe  -java

Current Depth: 7

[Initial Board State]

 ---------------------------------------
|    |    |    | WM |    | WM |    | WM |
|____|____|____|____|____|____|____|____|
| WM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | WM |    | WM |    | RM |    | WM |
|____|____|____|____|____|____|____|____|
|    |    | WM |    | WM |    | RM |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | WM |    | WM |    | RM |
|____|____|____|____|____|____|____|____|
| RM |    |    |    |    |    |    |    |
|____|____|____|____|____|____|____|____|
|    | RM |    | RM |    | RM |    | RM |
|____|____|____|____|____|____|____|____|
|    |    | RM |    |    |    | RM |    |
|____|____|____|____|____|____|____|____|
RED has 9 viable moves
of which 1 (optimal) moves will be considered.
```



```
of which 1 (optimal) moves will be considered.
__ __ __ WM __ WM __ WM
WM __ __ __ __ __ __ __
__ WM __ WM __ RM __ WM
__ __ WM __ WM __ RM __
__ RM __ WM __ WM __ RM
RM __ RM __ __ __ __ __
__ RM __ __ __ RM __ RM
__ __ RM __ __ __ RM __
```

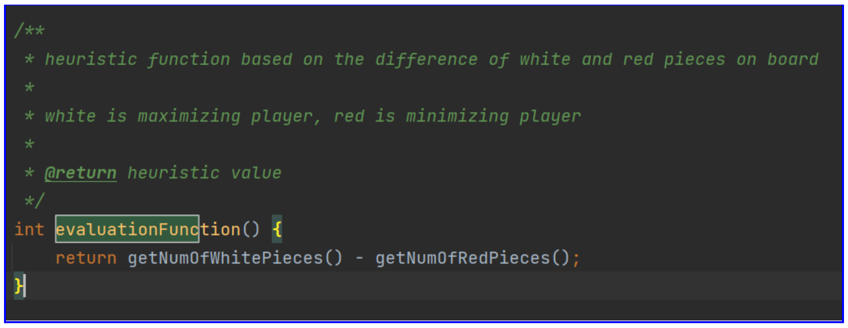
As we can observe, the number of moves the AI considers decreases as we increase the depth. However, what's interesting is that the puzzle solution differs from the solution my AI found most likely due to the fact that my heuristic of (difference in number of red and white pieces) is not that great.

Below are the run times I got from running different depths
(same position as above and same heuristic):

depth 0	run time:23ms
depth 1	 run time:31ms
depth 3	 run time:40ms
depth 5	 run time:72ms
depth 7	 run time:271ms
depth 9	 run time:932ms
depth 11	 run time:3176ms

As we can observe, the difference between the runtimes for each increase to depth get exponentially greater (e.g. difference between depth 1 and 3 was just 9ms, but was over 1000ms between depth 9 and 11).

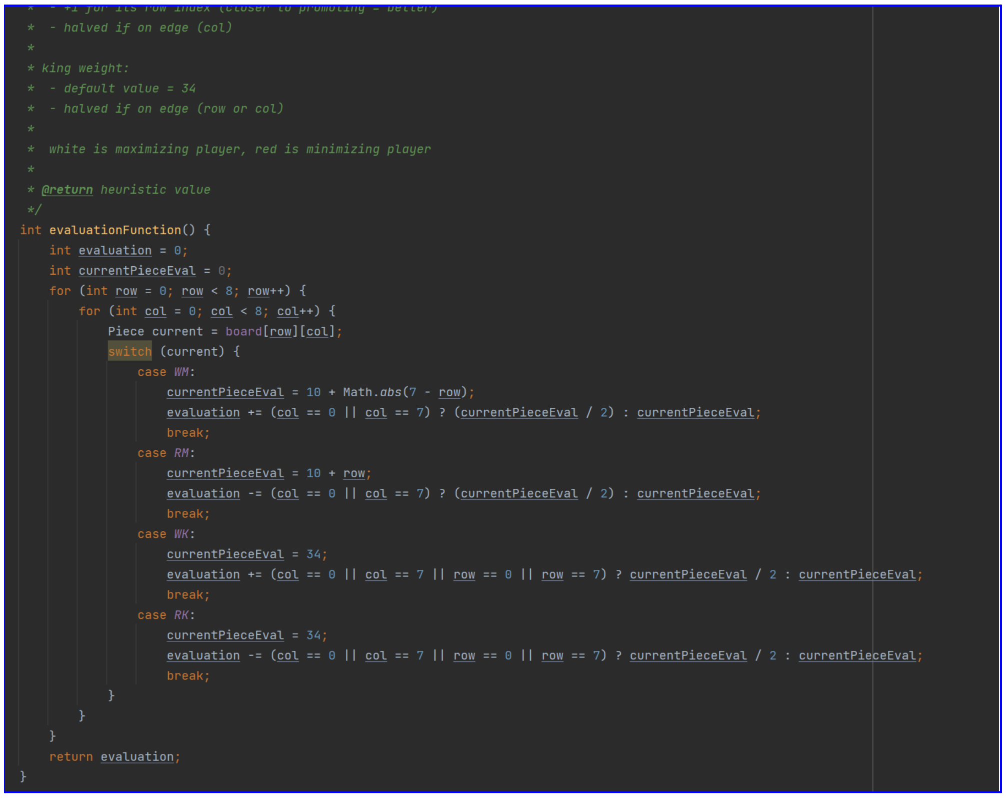
7. Implement at least two evaluation functions with varying quality. Compare the effect of improving the evaluation function.
My first evaluation function was simply the difference between the number of white and red pieces (indiscriminate of whether they were man or king pieces).



```
/**
 * heuristic function based on the difference of white and red pieces on board
 *
 * white is maximizing player, red is minimizing player
 *
 * @return heuristic value
 */
int evaluationFunction() {
    return getNumOfWhitePieces() - getNumOfRedPieces();
}
```

The evaluation wasn't bad per say, but sometimes it would do unintuitive moves like capturing a man piece opposed to a king piece given the same option. It also didn't prioritize capturing high risk enemy pieces (close to promoting).

My second evaluation function was more holistic. It considered the type of piece (man vs king), the level of threat (how close it was to promoting), and whether it was on the edge.



```java
 *  - +1 for its row index (closer to promoting = better)
 *  - halved if on edge (col)
 *
 * king weight:
 *  - default value = 34
 *  - halved if on edge (row or col)
 *
 *  white is maximizing player, red is minimizing player
 *
 * @return heuristic value
 */
int evaluationFunction() {
    int evaluation = 0;
    int currentPieceEval = 0;
    for (int row = 0; row < 8; row++) {
        for (int col = 0; col < 8; col++) {
            Piece current = board[row][col];
            switch (current) {
                case WM:
                    currentPieceEval = 10 + Math.abs(7 - row);
                    evaluation += (col == 0 || col == 7) ? (currentPieceEval / 2) : currentPieceEval;
                    break;
                case RM:
                    currentPieceEval = 10 + row;
                    evaluation -= (col == 0 || col == 7) ? (currentPieceEval / 2) : currentPieceEval;
                    break;
                case WK:
                    currentPieceEval = 34;
                    evaluation += (col == 0 || col == 7 || row == 0 || row == 7) ? currentPieceEval / 2 : currentPieceEval;
                    break;
                case RK:
                    currentPieceEval = 34;
                    evaluation -= (col == 0 || col == 7 || row == 0 || row == 7) ? currentPieceEval / 2 : currentPieceEval;
                    break;
            }
        }
    }
    return evaluation;
}
```

Man Piece weight:

- default: 10
- +1 for its row position on board (up to 6 because it promotes to a king on rows 0 and 7)
- halve its weight if on the edge (col 0 or col 7)

King Piece weight:

- default: 34
- halve its weight if on the edge (col 0 or col 7, row 0 or row 7)

In my (non-expert) opinion, I think a piece close to promotion is much more valuable than a piece at its starting position, however not twice as much. This was my reasoning for giving man pieces a default weight of 10 and plus 1 for its position.

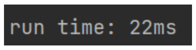
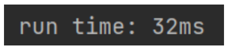
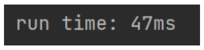
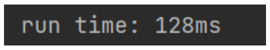
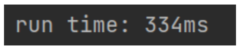
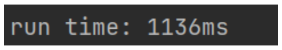
I also decided to halve a piece's weight if it was on the edge because you're effectively cutting its number of moves by half.

I weighed a king piece to be greater than a man because it can move both forwards and backwards. I also chose 34 because if it's on an edge, it's value is halved to 17 and the maximum value for a man piece can be 16 (10 + 6).

This is a roundabout way of saying that kings on an edge are worth more than a man that's about to be promoted (this is important because otherwise you would never promote a man on the 7th rank because it's worth more than a king on the 8th rank).

Evaluation function comparisons

Like the first evaluation function, the run time gets increasingly larger every time the depth is incremented. However, the run times themselves are pretty similar for both heuristics (same state was used).

depth 0	 `run time: 22ms`
depth 1	 `run time: 32ms`
depth 3	 `run time: 47ms`
depth 5	 `run time: 128ms`
depth 7	 `run time: 334ms`
depth 9	 `run time: 1136ms`

Sometimes the two evaluation functions would come up with a different number of optimal states for a given depth. Additionally, even if the number of "optimal" moves were the same at a given depth, they would sometimes pick different moves.