

Project 2: Checkers

Paul Beggs

[GitHub Link](#)

November 14, 2025

1 Evaluation Function

This basic evaluation function works by simply counting how many pieces each player has on the board, and then returning the difference between the current player's piece count and the opponent's piece count. Hence, it returns either a positive or negative integer, including zero if the counts are equal. The code snippet is provided in [Listing 1](#).

```
1 public class BasicEval implements ToIntFunction<Checkerboard> {
2     @Override
3     public int applyAsInt(Checkerboard value) {
4         PlayerColor current = value.getCurrentPlayer();
5         PlayerColor opponent = current.opponent();
6         int currentCount = value.numPiecesOf(current);
7         int opponentCount = value.numPiecesOf(opponent);
8         return currentCount - opponentCount;
9     }
10 }
```

Listing 1: Basic Evaluation Function

I've found that this evaluation function works well enough. It is capable of keeping their pieces from needlessly being captured, and capturing mine just fine. However, when it has an advantage, like being close to promoting their pieces, they don't press the advantage. For example, consider the board states from [Figure 1](#): Here, red can promote and have better pieces, but chooses to move the pieces closer to the back. I suppose moving to d3 also has merit, as they won the game either way.

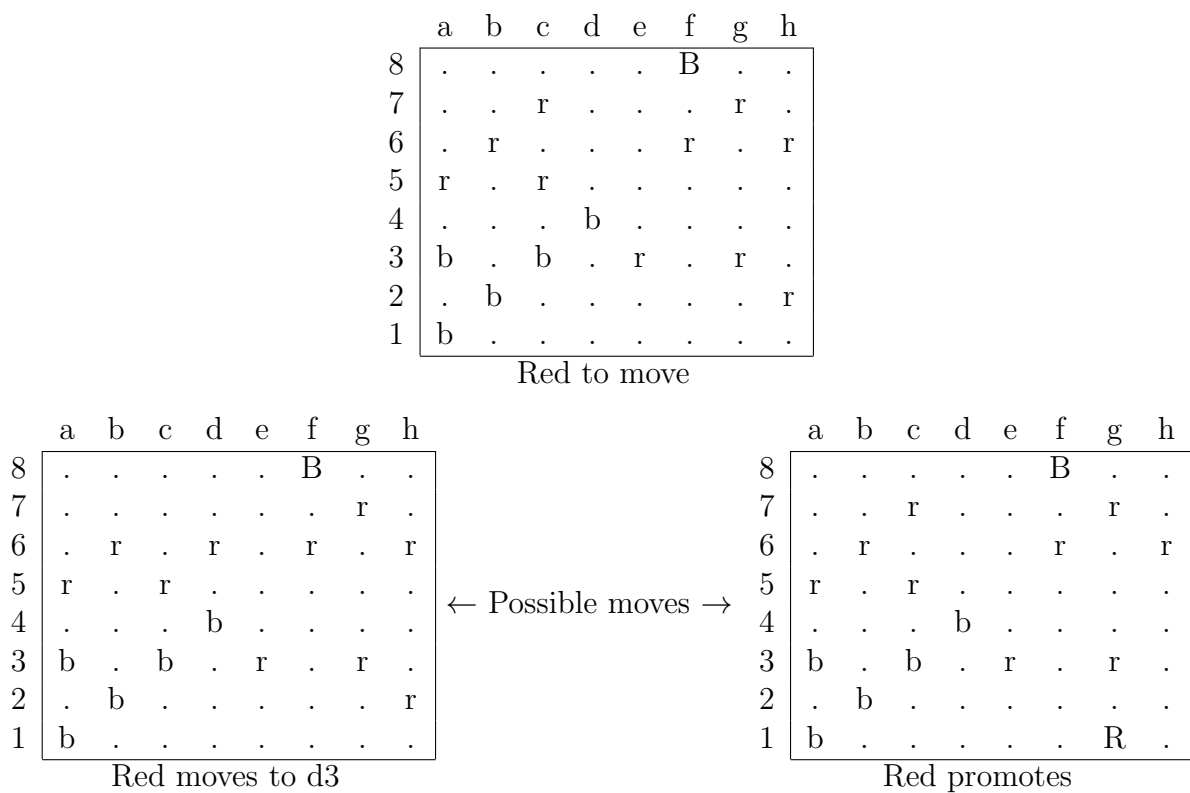


Figure 1: Board States

2 Autocheckers Results

By running experiments with different depth levels, I found that it has a major effect upon the chance of winning. For example, when Player1 had a depth limit of 7 and Player2 had depth limit of 4, Player1 won 18 times, and Player2 only won 3 times out of 32 games. That's a pretty fair margin in my opinion. I wasn't able to run larger depth limits > 8 , as the program took too long to run.