2ID90 International Draught assignment

Group 19: Daan de Graaf, Yoeri Poels

March 10, 2017

1 Introduction

Give a short introduction on your project: state the objectives and comment on how they are achieved. Further more, in the whole document, please take care of the following:

- *Refer to any material you used to obtain your results*, e.g. [**?**].
- *Illustrate your material with appropriate figures which are numbered and have a descriptive caption. Refer to the figure in the text: see* Figure 1.

Figure 1: The winners of the 2ID90 international draughts tournament, edition 2014.

- *Use pseudo-code to explain your algorithms. Note that pseudo-code is not the same as your actual java code. It should be an abstraction of that code and should be presented with readability and clarity in mind. Pseudo-code has to be accompanied with proper explanation and argumentation.*
- *Use numbered mathematical formulas instead of lengthy explanations in text. Additionally, discuss the principles behind the formula.*

2 Alpha-Beta

We made the decision early on to implement alpha-beta pruning in the single method below, in contrast to having two separate methods that are very similar in an effort to reduce code duplication. We added an extra boolean parameter 'maximize' to indicate whether the algorithm should minimize or maximize the evaluation function. We also added a 'depth' parameter to indicate how deep down the tree the algorithm may search. Upon a recursive call the 'maximize' parameter is flipped, and the 'depth' is decreased by 1. If the method is called with depth 0, the heuristic evaluation of the node is returned. We have added one more parameter 'didCapture', which is set to true only if the method was called recursively and in the given node state the previous move was a capture. This is only used to make sure that the best move of node is not set to a move that occurs after a capture, as captures do not count towards the depth. On top of this we also extend the search depth by two layers if the depth reaches 0 on a node that is not quiet. The resulting algorithm then becomes:

Algorithm 1 AlphaBeta

```
int AlphaBeta(node, alpha, beta, depth, maximize, didCapture) {
    // Search up to two extra nodes further if the current node is not quiet
    if ((depth <= 0 && isQuiet(node)) || depth <= -2)    { return Evaluate(node); }
    int bestValue = maximize ? ∞ : -∞;

    for (move : PossibleMoves(node) {
        node.doMove(move);  // Apply this move so we can evaluate the state
        int newDepth;
        if (move.isCapture()) {
            newDepth = depth; // Do not count captures in the search depth
        } else {
            newDepth = depth - 1;
        }

        int value = AlphaBeta(node, alpha, beta, newDepth, !maximize, m.isCapture());
        if (maximize && value >= bestValue) {
            bestValue = value;
            if (depth == searchDepth && !didCapture) {
                // We are at the root of the search tree
                node.setBestMove(move);
            }

            if (bestValue >= beta) {
                node.undoMove(move); // Undo move for clean return
                return bestValue;
            }
        }
        else if (!maximize && value <= bestValue) {
            bestValue = value;
            if (depth == searchDepth && !didCapture) {
                node.setBestMove(move);
            }

            if (bestValue <= alpha) {
                node.undoMove(move);
                return bestValue;
            }
        }
        node.undoMove(move); // Undo the move so we can reuse the node for the next iteration of the loop
    }
    return bestValue;

}
```

3 Iterative Deepening

We implement iterative deepening by first running Algorithm 1 with an initial search depth 'baseSearchDepth' and storing this move. Then while we still have time we keep repeating this process with an increasing search depth until the time runs out. When the time runs out the alphaBeta method throws an AIStoppedException, which is caught here and The algorithm is given by:

4 Evaluation

Our evaluation of board state s is split up in several parts, with each having a different weight. The total evaluation-score is based on an evaluation of the **amount of pieces and kings**, the **tempi**, **balance** and **coherence** of both sides, and the amount of moves that can be made.

The score for the amount of pieces/kings *piecesScore* is calculated in the following way:

$$piecesScore(s) = (\bigcirc(s) + 3 \cdot \bigcirc king(s)) - (\bullet(s) + 3 \cdot \bullet king(s)) \tag{1}$$

where for board state s, $\bigcirc$(s) and $\bullet$(s) are the number of white and black pieces, and $\bigcirc$king(s) and $\bullet$king(s) are the number of white and black kings, respectively.
We use this to evaluate since having more pieces / the opponents losing pieces means you are more likely to win, as you lose when you do not have any pieces left.

The score for the tempi of both sides *tempiScore* is calculated in the following way:

$$tempiScore(s) = whiteTempi(s) - blackTempi(s) \tag{2}$$

where *whiteTempi* and *blackTempi* are the following:

An example of whiteTempi (total value being 23) is shown in Figure 2:

Give a clear explanation of your evaluation function. Compare alternative evaluation strategies, alternative parameter settings, etcetera. Argue why you have chosen a particular evaluation function: make measurements

Algorithm 2 IterativeDeepening

```
1  Move IterativeDeepening(State s) {
       bestMove = null;
3      try {
           node = new Node(s.clone());
5
           // Set to initial search depth
7          searchDepth = baseSearchDepth;
           // Loop while not interrupted
9          while (true) {
               bestValue = alphaBeta(node, −∞, ∞, searchDepth, s.isWhiteToMove(), false);
11             bestMove = node.getBestMove();
               if ((bestValue == MAX_VALUE && s.isWhiteToMove())
13                 || (bestValue == MIN_VALUE && !s.isWhiteToMove())) {
                   // We have a winning strategy
15                 break;
               }
17
               searchDepth++; // Increase the search depth for the next iteration
19         }

21     } catch (AIStoppedException ex) {
               // Our time is up, return the best move according to the highest search depth reached
23             return bestMove;
       }
25 }
```

```
1      int whiteTempi(State s) {
           whiteTempi = 0;
3          for (every white piece in state s)
               whiteTempi = whiteTempi + distance from white piece to bottom row (in number of rows);
5          }
           return whiteTempi;
7      }

9      int blackTempi(State s) {
           blackTempi = 0;
11         for (every black piece in state s)
               blackTempi = blackTempi + distance from black piece to top row (in number of rows);
13         }
           return blackTempi;
15     }
```

and use graphs and tables where necessary. Use formulas (like (3)*) or pseudo-code:*

$$H(s) = \begin{cases} \infty & \bigcirc(s) = 0 \\ -\infty & \bullet(s) = 0 \\ \bigcirc(s) - \bullet(s) & otherwise \end{cases}, \tag{3}$$

where for board state s, $\bigcirc(s)$ and $\bullet(s)$ are the number of white and black pieces, respectivily.

5 Custom extensions

We have handled quiescence in Algorithm 1, where at depth 0 we also check if the board state is currently quiet. If it is we return the heuristic evaluation of the node, otherwise we allow the search to continue up to 2 levels further in the tree. to check if a state is quiet we use the following simple algorithm:

We also ignore obliged moves in the search depth, as can be seen on line 9-10 of Algorithm. 1.

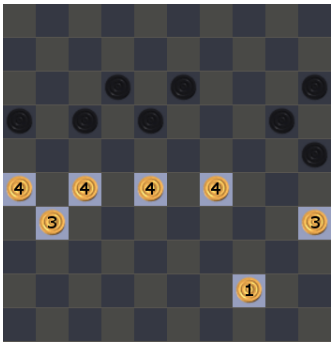

Figure 2: An example of whiteTempi

Algorithm 3 IsQuiet

```
Move IsQuiet(State s) {
        moves = s.getMoves();
        if (moves.isEmpty()) {
                // No pieces are left on the board
                return true;
            }
        return !moves.get(0).isCapture();
}
```

These extensions did make the AI stronger, but only slightly. When an otherwise identical version without these modifications (A) plays against the version with these extensions (B), A will draw B when A is white. But when B is white, it will win by a slight edge it has with the extensions. This is most visible in the end game:

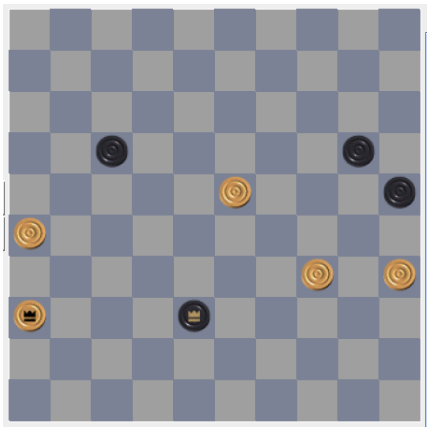

Figure 3: The game is nearly tied in piece count and the two kings suggest a draw outcome

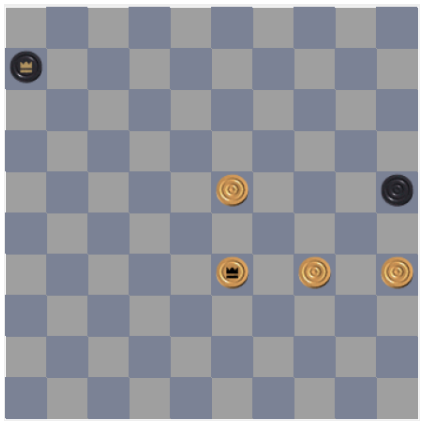

Figure 4: A is to move and has to strike the king and another piece, allowing B to capture the king and force A to put its last piece in harms way.

6 Results

If you did not do so already in the previous sections, you can show your final results here. Again you can use graphs and tables to show your point.

7 Conclusions

A short logical summing up of the main reported results.

8 Contributions

A statement on the contributions of each of the authors.

	implementation	**documentation**	**total #hours**
Author 1	60%	30%	30
Author 2	40%	70%	25

- *At least the given columns in the table need to be filled in, add columns if needed.*
- *Add comments to clarify your table entries when necessary.*
- *...*

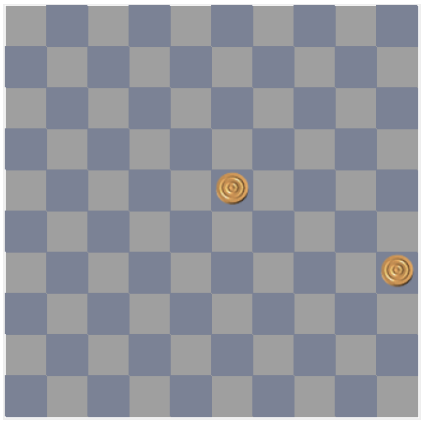

Figure 5: B wins by exploring further