

Assignment: Game Search

2ID90, edition 2015-2016 Q3

1 Overview

In this assignment you are going to build a computer player for the game of INTERNATIONAL DRAUGHTS. In short¹, International Draughts is a two player strategy game played on a 10x10 board of which only the dark fields are used to initially position 20 white pieces and 20 black pieces. The players in turn move a piece of their own color diagonally over the board. The white player starts the game. Pieces of the opponent may be captured by jumping over them to an empty field². Finally, the player without any remaining moves loses the game. See figure 1 for a picture of a draughts board in the initial position. You are challenged to write a draughts playing agent that will compete in a small competition against agents made by your fellow students. The overall best playing agent will play a game against a former Dutch champion. In global terms, you need to

1. write an alpha-beta search algorithm (Section 2),
2. write a function that can evaluate a state of the draughts game (Section 3), and
3. integrate these two in a framework for playing a (computer) draughts tournament (Section 4 and Section 5).

For this to work out you are given some functionality specific to the international draughts game (Section 6).

2 Alpha-beta search

For an ALPHA-BETA algorithm the following classes/interfaces are useful:

- class GameState with methods:
 1. getMoves(): for getting all valid moves in this state,
 2. doMove(Move): for making a move,
 3. undoMove(Move): for undoing a move.
- class Move, a class encapsulating the data for doing a valid move in a certain state.
- class GameNode³ with methods
 1. getGameState(): to return the game state of this node, and
 2. setbestMove(Move) and getBestMove(): to set or get the best move found for the game state of this node.

A rather simple and surely incomplete sketch of an alpha-beta algorithm then becomes something like this:

¹ If this game is unfamiliar to you you can find a full explanation of its relatively simple rules at [wikipedia](https://en.wikipedia.org/wiki/International_draughts).

² A player who can capture a piece, is *obliged* to do so.



Figure 1: An international draughts board with pieces in initial position.

³ This is the only class of the three classes mentioned here that you have to implement yourself when you start making your own computer draughts player.

```

1 int alphaBeta(GameNode node, int alpha, int beta) {
    GameState state = node.getState();
3    List<Move> moves = state().getMoves();
    for (Move move : moves) {
5        state.doMove(move);
        ... // recursive call
7        state.undoMove(move);
    }
9    node.setBestMove(bestMove);
    ...

```

It should be possible to stop an alpha-beta method call that takes too much time⁴. Technically, this boils down to throwing an exception, after a call to the stop() method sets a boolean to *true*:

```

private boolean stopped = false;
2 public void stop() { stopped = true; }

4 int alphaBeta(GameNode node, int alpha, int beta)
    throws AISToppedException {
6     if (stopped) {
        stopped = false;
8         throw new AISToppedException();
    }
10    ...

```

⁴ For fair play in the Contest environment of section 4, it is required that the computer players use the same amount of time.

3 Draughts Evaluation function

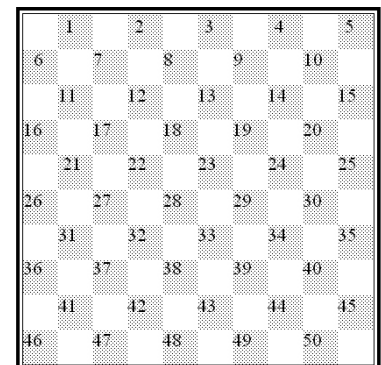
In leaf nodes of the game tree the state of the game has to be evaluated. To implement an evaluation the state of the game has to be inspected. A simple evaluation might count the number of pieces on the board. A skeleton of an evaluation function is given below. Note that the index of the pieces array starts at index 1; this is conform the International Draughts field numbering that goes from 1 to 50 as shown in figure 2.

```

import static nl.tue.s2id90.draughts.DraughtsState.*;
2 ...
int evaluate(DraughtsState ds) {
4     // obtain pieces array
    int[] pieces = ds.getPieces();
6     // compute a value for this state, e.g.
    // by comparing p[i] to WHITEPIECE, WHITEKING, etc
    ...
8     return computedValue;
10 }

```

Note that the gametree may have two types of leaf nodes: 1) actual leaves without successor states, and 2) nodes at which the maximal search depth is reached. These two nodes need a different treatment.



4 Tournament framework

As you are writing your own AI player you may want to test the player in the AICOMPETITION tool against other players or against a human player. See figure 3. In order to do so you need to implement the Player interface by inheriting from the DraughtsPlayer

Figure 2: The field numbers used on a draughts board. Initially, the white pieces are on the fields 31 to 50 and the black pieces on the fields 1 to 20.

class. From the Player interface you need to implement the `getMove(DraughtsState)` method, that should return a valid move in the argument game state. Furthermore, a `stop` method has to be implemented, similarly to the method discussed in section 2. For a player that plays random moves this is very simple as shown in the code below, where the `stop` method is empty, because the `getMove` function is almost instantaneous.

```

public class RandomDraughtsPlayer extends DraughtsPlayer {
    public Move getMove(DraughtsState s) {
        List<Move> moves = s.getMoves();
        Collections.shuffle(moves);
        return moves.get(0);
    }
    public Integer getValue() { return 0; }
    public void stop() { }
}
    
```

In your own implementation of a strong `DraughtsPlayer` the `getMove` method should contain a call to alpha-beta.

5 Installation

Two NETBEANS projects and a library folder are provided:

- AICompetition: a project for the AICompetition tool,
- 2ID90-Group-NN: a project containing an example Draughts player, and
- a lib folder with some additional libraries.

Branding the player project by replacing NN at several places by XY, *your group's number in oase*⁵, gives you your own player project:

1. rename the project 2ID90-Group-NN to 2ID90-Group-XY⁶.
2. In the src folder rename package `nl.tue.s2id90.groupNN` to `nl.tue.s2id90.groupXY`⁶.
3. In the arguments options⁷ of the AICompetition project, write `../2ID90-Group-XY/dist/ dist`.

After first compiling the AICompetition project and then your 2ID90-Group-XY project, you can run the AICompetition tool. In that tool pressing the `CREATE-SCHEDULE` button should give you a popup similar to that in figure 4.

6 International Draughts Library

The java library file `AICompetition.jar` contains several classes for International Draughts. Below the source code of some of them are given. The relevant classes and their source code and javadoc are provided.

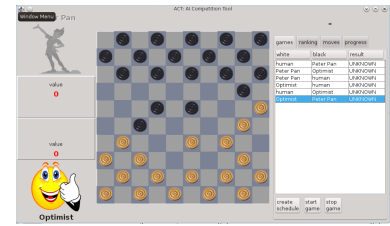


Figure 3: A screenshot of the AICompetition tool.

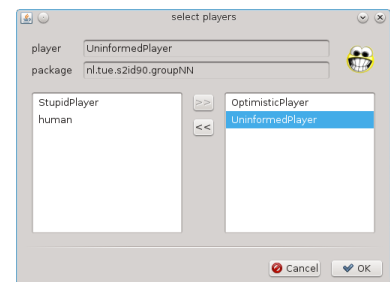


Figure 4: Popup of the AICompetition tool for selecting the players in a contest.

⁵ One group member should register in oase to claim a group number.

⁶ Do this using the NetBeans functionality for renaming projects/packages, available via the context menu of the project/package node in the projects list.

⁷ The command line options are specified in the run tab of the project properties.

6.1 Gamestate

This interface is implemented by the DraughtsState class.

```

1 package nl.tue.s2id90.game;
import java.util.List;

3 /** class that keeps the state information of a Game. */
5 public interface GameState<Move> {
    /** @return a list of valid moves in this state. */
    List<Move> getMoves();

    /** @return true iff valid moves in this state exist. */
    boolean isEndState();

    /** applies move m in this state. */
    void doMove(Move m);

    /** undoes the effect of move m. */
    void undoMove(Move m);

    /** @return whether or not the white player is to move. */
    boolean isWhiteToMove();

    /** resets state to initial game state. */
    void reset();
23 }

```

6.2 Player interface

This interface is implemented by the DraughtsPlayer class. Implementations of the latter class may be provided to the AICompetition tool where they can be selected for playing a game.

```

1 package nl.tue.s2id90.game;
import nl.tue.win.competition.util.Identity;
3 public interface Player<Move, State> extends GameState<Move>>
    extends Identity {
    /** computes a valid move in the given game state s.
     * This method should be prepared
     * to immediately return a Move when stop() has been called.
     * @return a valid move in State s.
     */
    Move getMove(State s);

    /** @return the computed value of the last Move.
     * Returns null if that value is not available.
     */
    Integer getValue();

    /** as a result of this call the Player should immediately
     * return a value in getMove().
     */
    void stop();
21 }

```

7 What to do next?

To come to a basic draughts player you need to do the following.

- Implement alpha-beta.
- Implement an initial evaluation function that just counts pieces.
- Implement stopping
 - throw an exception when stop has been called
 - implement Iterative Deepening to always have a result.

From here on you can repeatedly try to improve your player in several ways, for example:

- enhance evaluation function.
- improve search function.
- improve stop criterion for searching.
- reuse results in a previous iteration of Iterative Deepening.

Improvements that hinder your opponent are not allowed⁸.

⁸ So, it is not allowed to search while the opponent is searching; it is not allowed to keep results between two calls to `Player::getMove()`.

8 Appendix

8.1 Tournament rules

1. A computer player that makes an illegal move loses the match immediately.
2. The number of seconds per move is unknown to a computer player: A computer player loses a match immediately, if a move is not supplied within a reasonable amount of time after receiving a call to its stop method.
3. Decisions of the lecturers of 2ID90 always prevail and are beyond discussion.
4. The tournament consists of one or more GROUP phases followed by a KNOCK-OUT phase.
5. In a group phase there are X groups of maximum Y teams. In each group a single round-robin tournament is played, i.e. all teams play each other once in a match. In each match a total two points can be earned: A winning team gets 2 points, in case of a tie each team gets 1 point. For each group the team that collects most points goes to the next phase. In case of a tie, the team with the most points in matches between the tied teams is the winner. If this is insufficient, the drawing of lots will determine the winner.
6. The knock-out phase consists of several final rounds, e.g. 4 quarter finals, 2 semi finals, and a final. If matches in these finals are a draw, a rematch is played. If the rematch is a draw, the winner is determined by drawing of lots. For the finals, if time permits an additional rematch may be played instead.