NAVIGATION

B4B36ZUI

tasks

- Task 1 - EN - Path Planning
- Task 1 - CS - Path Planning
- Task 2 - EN - Nonograms
- Task 2 - CS - Malované křížovky
- Task 3 - EN - Game Theory - Gobblet
- Task 4 - EN - Monkey and banana
- Task 5 - EN - Grid-world

Přednášky / Lectures 2018

Cvičení / Labs 2018

Introduction to Artificial Intelligence (B4B36ZUI,BE4B36ZUI)

ALL COURSES

Summer 2017 / 2018

Older

Task 3 - EN - Game Theory - Gobblet

(description of the game from boardgamegeek.com)

Gobblet is an abstract game played on a 4×4 grid with each of the two players having twelve pieces that can nest on top of one another to create three stacks of four pieces.

Your goal in Gobblet is to place four of your pieces in a horizontal, vertical or diagonal row. Your pieces start nested off the board. On a turn, you either play one exposed piece from your three off-the-board piles or move one piece on the board to any other spot on the board where it fits. A larger piece can cover any smaller piece. A piece being played from off the board may not cover an opponent's piece unless it's in a row where your opponent has three of his color.

Your memory is tested as you try to remember which color one of your larger pieces is covering before you move it. As soon as a player has four like-colored pieces in a row, he wins — except in one case: If you lift your piece and reveal an opponent's piece that finishes a four-in-a-row, you don't immediately lose; you can't return the piece to its starting location, but if you can place it over one of the opponent's three other pieces in that row, the game continues.

Assignment

Input: You will be given two classes that define the game Gobblet – `Board.java` that contains the representation of the states, generates possible moves for a given state, provides heuristic evaluation function for non-terminal states, etc. `Move.java` is a class that defines moves in the game. Using these two classes, you are able to implement an algorithm for solving Gobblet games and finding optimal strategies.

The goal of the assignment is to implement Alpha-Beta pruning algorithm or one of its variants (e.g., Negascout). Your algorithm will be given a state and desired number of moves the players can execute (depth), and the task of the algorithm is to compute **value of the game** given these parameters and using the evaluation function specified in `Board.java` .

The main goal is to create as effective algorithm as possible. The effectiveness of the algorithm is primarily measured by the number of evaluated nodes – nodes that the algorithm opens. Note, that for Negascout this number includes repeated visits.

Hints

1. Do not create your own heuristic evaluation functions for Gobblet – this is not the point. Use the resources that you already have.
2. Recall what factors can speed-up Alpha-Beta algorithm and implement such improvements using available methods from Board.java

Output: Value of the game. (Your algorithm is given a state and a depth and returns the value of the game).

Quality criteria number of expanded nodes

Grading

1. Algorithm fulfils the assignment completely.
2. The value of the game must be the same as the value of the reference solution.
3. The quality of the solution is given by the number of expanded nodes (it's implemented in the `Algorithm` class by method `getNodesCount()`). Lower is better.
4. Our reference solutions have 3 levels of quality with score weights 2, 3, 3, respectively. You are awarded the points if you beat them.
5. There are several test experiments in BRUTE, the score is summed up over these experiments and normalized to maximum score 8.

Submission

- **Date/time**: 22/04/2018 23:59:59
- **Content**: compressed content of Java package `cz.cvut.student` (pack only .java source files, no subfolders please)

Implementation details

- Use the provided codebase: gt-student-package.zip.

You are given a runnable main class `Gobblet` :

- there are 3 basic configurations (parameters seed, randomMoves, depth) you can you use to test your algorithm.
- the game is constructed with these parameters, the game board is constructed and several random moves are played to reach your initial state.
- Your task is to compute the value of the game in the initial state for the given depth. Your algorithm is called by method `run` .
- The quality of your algorithm increases with the decreasing number of nodes visited.

Your contribution comes in `AlphaBeta` class

- Implement class `AlphaBeta` in package `cz.cvut.student` (implementation of the AbstractClass `Algorithm`)
- In `AlphaBeta` class, there is an example usage of the methods needed for your implementation.
- Modify the class, mainly method `runImplementation` so it works as desired! Make sure to call `run` method for recursive call, **not the** `runImplementation` **directly**!
- Use default `AlphaBeta()` constructor only.

In you code you must call `run` method, **not** the `runImplementation` !

- Have fun! Don't try to avoid fair counting of expanded nodes, it's cheating!