DM 537 Object-Oriented Programming

Fall 2013 Project (Part 2)

Department of Mathematics and Computer Science
University of Southern Denmark

November 21, 2013

Introduction
The purpose of the project for DM537 is to try in practice the use of programming techniques and knowledge about the programming language Java on small but interesting examples.

The project consists of two parts.

Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules
This second part of the project is a part of the final exam. Both parts of the project have to be passed to pass the course.

Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Deliverables
A short project report (at least 4 pages without appendix) has to be delivered. This report has to contain the following 7 sections:

- **front page** (course number, name, section, date of birth)
- **specification** (what the ADT should be able to do)
- **design** (how to represent the ADT data and operations)
- **implementation** (how the ADT was implemented in Java)
- **testing** (what tests you performed)
- **conclusion** (how satisfying the result is)
- **appendix** (complete source code)

The report has to be delivered as a single PDF file electronically using Blackboard's SDU Assignment functionality.

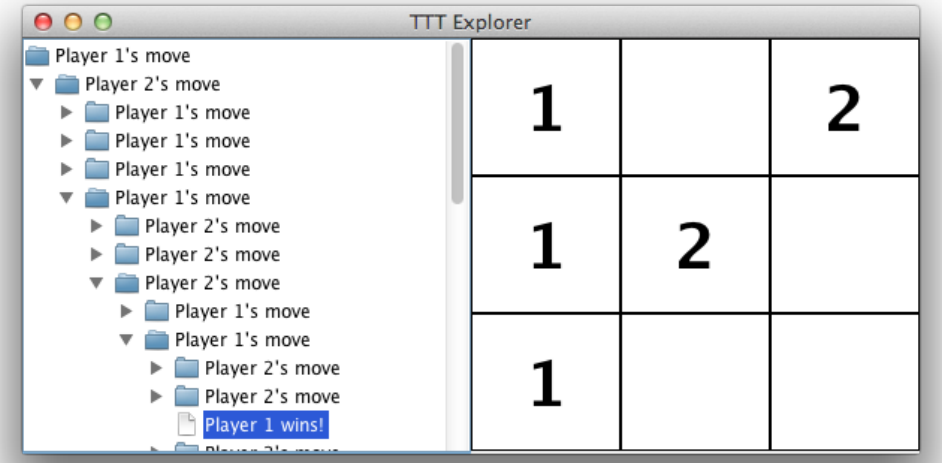
Deadline

Friday, January 10, 23:59

Project "Board Games: Tic Tac Toe & Co"

A game tree is a tree, where each node is representing a state of the game and a node is child of another node if, and only if, its state is the direct successor of that node. The root of the tree is the initial game state, i.e., where no player has yet made a move. The children of that root state are the states resulting from all possible moves that Player 1 could have made. The children of those children are likewise the states resulting from all possible answering moves that Player 2 could have made.

For the second part of the project, we again consider n-way Tic-Tac-Toe where n players play on a $(n+1) \times (n+1)$ grid and the first player to put 3 marks in a row, column, or diagonal wins. Our goal is to construct and visualize the complete game tree for n-way Tic-Tac-Toe. The screenshot below shows a part of that game tree, up to a node, where Player 1 wins.



Task 0 – Preparation

On the course home page, you find a directory with a number of changed (`Game.java`, `GUI.java`, and `TTTGame.java`) and new Java classes (`GameTree.java`, `GameTreeDisplay.java`, `GUIPanel.java`, `TTTExplorer`). Download all these classes to a new directory separate from your first part of the project.

Then download the unchanged classes (`CLI.java`, `TicTacToe.java`, and `UserInterface.java`) from the first part of the project to the same directory.

Finally, copy your completed classes (`Coordinate.java` and `TTTBoard.java`) from your solution for the first part of the project to the new directory.

Test those files by compiling and running `TicTacToe.java`. Fix the bug in the copying constructor of `TTTBoard.java` by adding the line `this.board = new int[this.size][this.size];` before the two nested for loops.

Task 1 – Implement ADT
Your first task is to specify, design, and implement the ADT for Tic-Tac-Toe game trees. Make sure that you satisfy the three following conditions:

- The class whose instances represent game trees is called `TTTGameTree`.
- Make sure that `TTTGameTree` implements the `GameTree` interface. This is required for your ADT to be compatible with the `GameTreeDisplay` user interface used by the main class `TTTExplorer`.
- The class `TTTGameTree` has a constructor that takes the number of players and constructs an initial game tree with just the initial game represented as the root of the tree.

Test your ADT by adding an empty implementation of the method `public void expand()` and running `TTTExplorer`.

Task 2 – Building the Game Tree
Your next task is to build a fully functioning method `public void expand()`, that will iteratively or recursively add children to the root node and its descendants until a winning state or a draw is reached. The result should be the full game tree.

Test your expansion by running `TTTExplorer` and analyzing the game tree by hand. Correct any mistakes that you find until your game tree contains all possible games. You might need to give more RAM to the Java Runtime Environment. This is usually done by adding the parameter "`-Xmx2048m`" after "`java`" for a maximum of 2 GByte of RAM.

One possible approach is to use a `java.util.Queue` to store the nodes that still have to be visited. Initially, this is just the root node. Then, while the queue still has some elements, the first element is retrieved using the `poll` method. All successor game states for that node are computed and added as children of that node. These new nodes are then added to the `Queue` object, such that they will be handled in a later iteration. This implements an iterative breadth-first search.

Task 3* – Reducing the Size of the Game Tree
Your first challenge task is to reduce the memory footprint of the game tree. To this end, in the `public void expand()` method you should keep a `java.util.Map` from games (instances of `TTTGame`) to nodes of your tree. Whenever you create a new node, add a mapping from its game to the node to this map. Before creating a new node, look if you already have a node for the current game. If so, reuse that node. In this way, the memory footprint can be reduced by orders of magnitude. For this to work, you have to implement `public int hashCode()` and `public boolean equals(Object obj)` for the `TTTGame` class. Remember that `hashCode` needs to return identical numbers for game states that are considered equal according to `equals`.

You can also try to reduce the number of game states by considering states that are equal with respect to rotation and/or flipping only once. For this, you will have to at least adapt your `hashCode` and `equals` methods.

Note that this task is optional and does not have to be solved for this part of the project to be considered as passed.

Task 4* – Artificial Intelligence
Your second challenge task is to modify the Tic-Tac-Toe implementation in such a way that a game tree is used to implement an AI player to play against the human player. I.e., when starting `TicTacToe`, first the game tree is computed. Then the human player and the AI player take turns making moves. The AI player uses the game tree in any situation to make moves that give it the best possible result (win or draw, if possible from the current state). You can also implement two or more AI players, such that the computer can play against itself.

Note that this task is optional and does not have to be solved for this part of the project to be considered as passed.