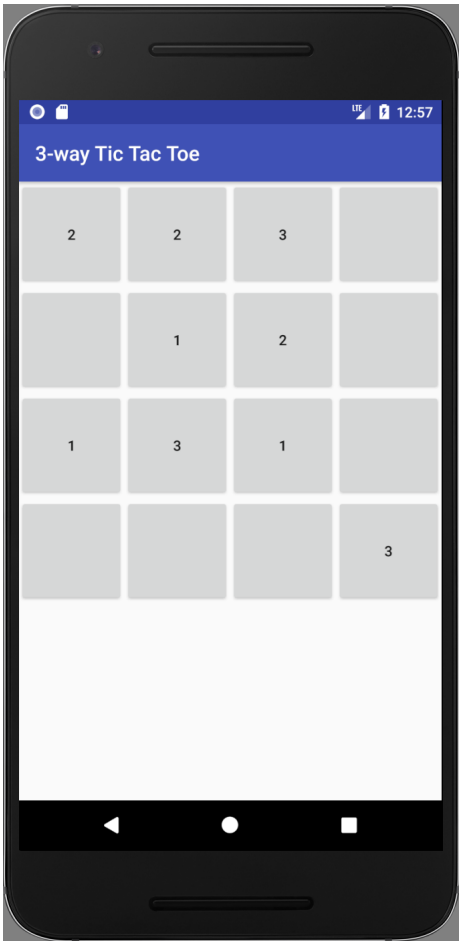
DM550/DM857
Introduction to Programming
Fall 2017 Project (Java)

Department of Mathematics and Computer Science
University of Southern Denmark

November 20, 2017



Introduction
The purpose of this project is to try in practice the use of object-oriemnted programming techniques as well as knowledge about the programming language Java. Please make sure to read this entire note before starting your work on this project. Pay close attention to the three sections below.

Exam Rules
This project is part of the final exam for DM550 and DM857. Both the project qualification assessments, the Python project, and this Java project have to be passed in order to pass the overall exam. The project should be done in groups of 3 students. Other group sizes require written confirmation.

Deliverables
A project report (10-20 pages without appendix) containing the following 6 sections has to be delivered:

- **front page** (course number, names, sections, dates of birth)
- **specification** (what the program is supposed to do)
- **design** (how the program was planned)
- **implementation** (how the program was written)
- **testing** (what tests you performed)
- **conclusion** (how satisfying the result is)

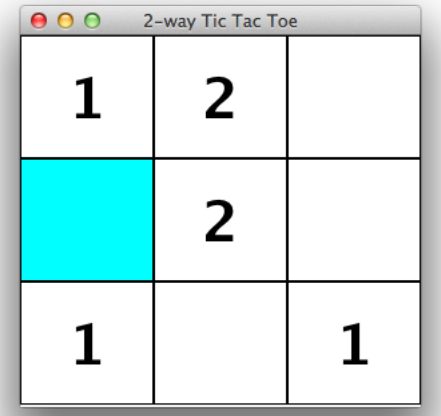
The report has to be delivered as a PDF file (suffix .pdf) together with the Java source code files (suffix .java) and any needed supporting data files electronically using Blackboard's SDU Assignment functionality. In addition, one printed copy has to be delivered to the "dueslag" of the respective teaching assistant. **Do not forget to include the complete source code!**

Deadline
The deadline for this project is

Friday, January 12, 23:59.

Late deliveries cannot be accepted, so please plan your time accordingly.

n-way Tic Tac Toe



Tic-Tac-Toe is a simple board game traditionally played by 2 players on a 3×3 grid. The players alternate in placing a mark on one of unmarked fields. A player wins as soon as any of the rows, columns, or diagonals contain 3 of his or her marks in a row. The game is a draw, if no player wins and all 9 fields have been marked.

Here, we consider the extension to n-way Tic-Tac-Toe where n players play on a $(n + 1) \times (n + 1)$ grid. The winning condition is the same, i.e., the first player to put 3 marks in a row, column, or diagonal wins.

For more information, on the standard variant, consider the Wikipedia entry: `http://en.wikipedia.org/wiki/Tic-tac-toe`

Task 0 – Preparation

On the course home page, you find a directory with a number of Java classes (`AppUI.java`, `CLI.java`, `Coordinate.jav`, `GUI.java`, `Game.java`, `ScrollView2D.java`, `TTTGame.java`, `UserInterface.java`) as well as two classes (`TTTBoard.java` and `XYCoordinate.java`) that are incomplete.

Your task is to download these files and put them into a directory for your project. If you are going to make an app, there is a zipped Android Studio available for your convenience (`TicTacToe.zip`). Read all files and look up unknown classes and concepts in the Java API documentation (`http://download.oracle.com/javase/7/docs/api/`), the Android reference (`https://developer.android.com/s`), in the course book, and in the supplementary course book. Make sure you understand what is going on here in detail!

Task 1 – Bounding and Shifting Coordinates
Take a closer look at the incomplete methods in `XYCoordinate.java`:

- `checkBoundaries` needs to check whether the position represented by this object of class `Coordinate` is on a board of width `xSize` and height `ySize`, where the top-left corner has the position (0,0) and the bottom-right corner has the position (`xSize`-1, `ySize`-1). In the case of a 3×3 grid, the valid positions are (0,0), (1,0), (2,0), (0,1), (1,1), (2,1), (0,2), (1,2), and (2,2). Here, for example, the top-right corner is (2,0).s

- The method `shift` needs to construct a new object of class `Coordinate` with coordinates shifted `dx` to the right and `dy` down. For example, calling `pos.shift(-1,1)` with `pos` representing the position (1,1) would result in a new position (0,2).

Task 2 – Implementing The Board
Your next task is to implement the missing methods in `TTTBoard.java`:

- `isFree` needs to return true if, and only if, the given position is free, i.e., if a value of 0 is saved in the appropriate cell of the array representing the board.

- `getPlayer` needs to return the number of the player that made a move at the given position or 0, if the position is free.

- `addMove` needs to first check that the position is actually on the board (Hint: use `checkBoundaries`) and that the given player number is valid.. If not, an `IllegalArgumentException` should be thrown. If the given position and player number are valid, the position should be marked by the player number.

- `checkFull` needs to return true if, and only if, there are no more free positions on the board.

- `checkWinning` needs to return 0 if no player has won yet. Otherwise, it should return the number of the player who has 3 in a row, column, or diagonal. Make sure that your code does not only work for 3×3, but also for quadratic grids of any size!

  You can, if you want, simplify the implementation of `checkWinning` by implementing the helper method `checkSequence` that checks rows (`dx == 1`, `dy == 0`), a columns (`dx == 0`, `dy == 1`), or diagonals (`dx == 1`, `dy == 1` *or* `dx == 1`, `dy == -1`) from a given start position.

Task 3 – Testing the Game
There are three user interfaces for the game: an Android app one (`AppUI.java`), a Swing one (`GUI.java`), and a command line-based one (`CLI.java`).

Test at least one (and preferably multiple) of these three with your implementations from Tasks 1 and 2 with both the standard 3x3 grid but also with at least one larger board. Document the results of your testing and fix possible errors in `XYCoordinate.java` and `TTTBoard.java`. It is important to test at least the case of Player 1 winning, the case of Player 2 winning, and the case of the board being full with no player winning. In all cases, you should obtain a message indicating the result from the user interface.

Task 4 – Artificial Intelligence (challenge task) Implement an AI player that has an optimal strategy for at least 2-way Tic Tac Toe. That is, the AI player should always win or get a draw when it starts, and it should should never lose when the human player starts.

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

Beyond Tic Tac Toe

After completing the Tic Tac Toe game you are to develop your own board game implementation. To pass the Java project you have to complete at least **one of the following three tasks** successfully. Note that they differ regarding their difficulty.

In other words, it is sufficient to just do Task 5. But if you choose to do either Task 6 or 7, you do not have to perform Task 5.

Task 5 – Connect Four (obligatory level – easiest)
A board game similar to Tic Tac Toe is Connect Four, where a move consists of dropping a discs as far down as possible in a given column. There are two players on a larger board (typically broader than high) and the goal is to have four of your discs in a row, column, or diagonal.

The main difference to Tic Tac Toe with respect to game play is, that when you click on a column, the mark is made in the lowest free position of that column. The main difference with respect to the user interface is that the board is rectangular rather than quadratic.

You find more information about Connect Four on the Wikipedia entry: `http://en.wikipedia.org/wiki/Connect_Four`

Your task is to implement two classes: `CFGame.java` and `CFBoard.java`. These two classes should implement the game logic of Connect Four, with `CFGame` being a subclass of `Game.java` and using `CFBoard.java`. You also need to adapt at least one of the three user interfaces to be able to play Connect Four. As always, document your code and document your tests.

Hint: You can either write the two new classes from scratch or start with a copy of `TTTGame.java` and `TTTBoard.java`, respectively.

Task 6 – Go (supplementary level – medium difficulty)
Even though it is played by two players with only one type of figures, Go is a rather complicated board. Go is typically played on a 19×19 grid. It is very rich in strategic challenges and can be played for many hours. You find more information about Go on the Wikipedia entry: `http://en.wikipedia.org/wiki/Go_%28game%29`

Your task is to implement and test Go by writing the two new classes `GoGame.java` and `GoBoard.java` and adapting one of the three user interfaces. The challenges include winning conditions based on scoring and the removal of figures according to a number of rules.

Task 7 – Puzzle Games (challenge level – hardest)
There are many intractable puzzle games, a lot of them either stemming from Japan or having been popularized there. Some of the more known are numerical puzzles such as Sudoku and Kakuro. More interesting and challenging ones include Masyu, Nurikabe, Kuromasu, and Fillomino. There are also some puzzles with a richer background story, for example Battleship and Sokoban. Information about all of these games is easily obtained from the relevant Wikipedia pages.

Your task is to choose, implement, and test one of the above games. Your implementation should contain a user interface, a generator for problems, the possibility to solve problems manually, and at least a rudimentary automated solver. If you want to take on another game this requires written approval.