OPI F2008: Object-Oriented Programming
Carsten Schürmann
Date: April 2, 2008

Project
Connect Four (Version 1.1)

Guidelines

While we acknowledge that beauty is in the eye of the beholder, you should nonetheless strive for elegance in your code. Not every program which runs deserves full credit. Make sure to state invariants in comments which are sometimes implicit in the informal presentation of a class. Do not reinvent wheels, and try to make your methods small and easy to understand. Use tasteful layout and avoid long winded and contorted code. You are expected to work alone, but of course, you can discuss problems, solutions and strategies with others.

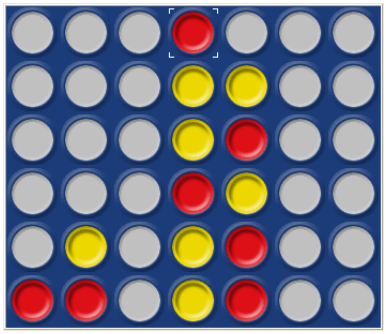
Goal

The goal of this project is to implement the game Connect Four. The project is designed as a four week project. You are asked to hand in two "homeworks", which will be graded. After you handed in your final implementation I will organize a Connect Four competition, where your programs will play against one another. The winner will receive a price.

Game Description

Connect Four is a two-player board game in which the players take turns in dropping tokens into a vertical grid with the objective of getting four of one's own tokens into a line. The game was published by Milton Bradley in 1974; a non-proprietary version is known as "The Captain's Mistress". See `http://en.wikipedia.org/wiki/Connect_Four` for more information.

In the game, there are two colors of tokens. One player plays yellow, the other red. The following depicts a game configurations that may occured in a concrete game.



If we enumerate the columns with 0..6, we see that if yellow plays 2 and red plays 2, red will have won, because red has managed to get four pieces into a line. This one is diagonal, but it could also be horizontal or vertical.

Rules

1. The game board, has seven columns and six rows.
2. There are 21 red and 21 yellow tokens.
3. One player plays with red tokens, the other with yellow tokens.
4. The tokens are inserted at the top of a column, and they will fall down and land on the ground (if the column was empty) or on top of a previously inserted token.
5. Red starts.
6. Red and yellow take turns.
7. One can only insert tokens in one of the seven columns.
8. One cannot insert a token into a column that is full.
9. A line consists of several tokens, either in vertical, horizontal, or diagonal form, which contain only tokens of the same color.
10. A player wins if he or she manages to form a line of four tokens of his or her color.
11. The game ends if one of the player wins.
12. The game ends in a tie if none of the two players can make a valid move and none of the players has won.
13. A move cannot take more than 18 seconds.

Project Goals

You are asked to implement two players for the Connect Four game and a graphical user interface. A human player and an automatic player. We will provide you with one interface that describes what a player is expected to do and you are supposed to implement this interface. No more rules are given.

One of the most critical steps in this part of this project is to think of a good representation of the game board. You are expected to design one that you consider most appropriate for your player, i.e. you can quickly make decision about if you won or lost, etc.. Then you are asked to implement an interface to the human player. We expect you to implement a graphical user interface. For the second assignment you are expected to hand in a program that allows two human players to play each other.

The automatic player is like a human player except that the computer uses an algorithm to determine where to put one the next token. This could be a search algorithm, just like the ones we discussed in class, depth first search, breadth first search, iterative deepening, or maybe even $\alpha\beta$-pruning search. (This will likely be my choice for my player!) We will not accept a player, that is based on the idea of randomly tossing a token into one of the columns. You are required to implement some kind of search algorithm.

Interfaces

Before telling you in detail what you have to do, we establish some common basic rules for the implementation. First, the color red and yellow are represented by Boolean values, where

red	is represented as `true`
yellow	is represented as `false` .

Every player you write, human or automatic, should implement the `Player` interface. Your automatic player needs to have an internal representation of the board, otherwise it will not be able to efficiently search for the best move. The human player doesn't do it, because we have memory, and are used to have the game situation in our heads. But this also means, that the automatic player needs to be informed about the move of the opponent. The human player observes that directly. Therefore, the `Player` interface that we ask you to implement the following four methods.

```
public interface Player {
    void init (Boolean color);
    String name ();
    int move ();
    void inform (int i);
}
```

`init` is the method that the Game referee (implemented and administered by the teaching staff) will use in order to inform a player about what color he/she/it is playing. The `method` serves a player to identify himself or herself for the logs. The `move` method is called when a player is asked to make a move, and the `inform` method is called to inform the player about the move of the opponent.

Code

Please find the sample code on the webpage. Your directory tree should look something like

```
. ---  Player     -- Player.java
   |
    -  <yourname> -- <yourname>.java
                     Board.java
                     GUI.java
                     ...
```

Where the `yourname` directory contains all of your code. *Please substitute your name for* `yourname`. You need to import the player inferface with `import Player.Player;`. It is important, that your classpath contains the root directory of that tree. (Denoted here by `.`) If you have questions, on how to set this up, please ask the TAs.

Problem 1

Hand in your design of a board, with all methods that you think are necessary and important. '

Problem 2

Hand in your implemenation of the human player and the graphical user interface!

Problem 3

Hand in your automatic player in *one* file (containing all of the classes that you need to run your automatic player) called `<yourname>.java`. The file should start with

```
/* Connect 4 */
/* My wonderful Connect4 Player implementation */

package <yourname>

import Player.Player;
....
```

Your file should provide a class called.

```
public class <yourname> implements Player {
...
}
```

For, example, my submission is called `CarstenSchuermann.java` providing the class

```
public class CarstenSchuermann implements Player {
...
}
```

Please follow these instructions carefully. Non-compliance will result in disqualification.