Board Game Client/Server package

Clement Gehring

February 13, 2014

1 Overview

This software consists of two packages, each implemented in Java:

1. boardgame: a generic board game client/server package
2. halma: an implementation of the halma game

The `boardgame` package is intended to allow the implementation of different board games and AI players with minimal effort. It provides networking support, log file management, and GUI support. No familiarity with this code is required to build an AI player, and only limited knowledge is required to implement an entire game.

To play, two clients (the players) connect to the server via TCP sockets. This allows the clients to be run on separate computers. A GUI which can be used to display an ongoing game or to examine existing log files is provided, but is not required to run a game. This software was built and tested using Sun JDK 1.6.0, but should be source-compatible with Sun JDK 1.5.

The source files are arranged in the following manner:

```
src
|-- boardgame
| |-- Board.java                       Abstract game board logic class
| |-- Move.java                        Abstract game move class
| |-- Player.java                      Abstract player class
| |-- Client.java                      Generic client, implements networking
| |-- Server.java                      Generic server, implements networking
| |-- HumanPlayer.java                 Generic remote GUI client
| |-- ServerGUI.java                   Main GUI class
| \-- BoardPanel.java                  Displays and gets input for a board in GUI
\-- halma
  |-- CCBoard.java                     Specific game logic
  |-- CCMove.java Move                 formatting for TCP transport
  |-- CCBoardPanel.java                Custom display for GUI
  |-- CCHumanPlayer.java               Remote GUI client
  \-- CCRandomPlayer.java              Random player
```

2 Launching the Server

To start the server, trype:

```
java -cp ./path/to/projectsrc.jar boardgame.Server [-p port] [-ng] [-q] [-t n] [-b class]
```

where:

- `-p port` sets the TCP port to listen on. (default=8123)
- `-ng` suppresses display of the GUI.
- `-q` indicates not to dump log to console.
- `-t n` sets the timeout to n milliseconds. (default=1000)
- `-b class` determines the game to run. (default=halma.CCBoard)
- `-k` indicates to start a new server once a game is running.

For example, the command:

```
java -cp ./path/to/projectsrc.jar boardgame.Server -p 8123 -t 300000 -b halma.CCBoard
```

launches a server on the default TCP port, with the GUI, a timeout of 300 seconds, and running the Halma game.

The server waits for two clients to connect and does not accept any more connections unless the -k flag is passed as an argument. Closing the GUI window will not terminate the server. The server exits once the game is finished. Log files, which record all of the moves in each game, are automatically written to the logs subdirectory. The log files contain a list of all moves, as well as information about the game's class and other parameters. The server also maintains a file, outcomes.txt, which keeps a summary of game results. At present this consists of the integer game sequence number, the name of each player, the color and name of the winning player, the number of moves, and the name of the log file.

3 Launching a Client

The server waits for four clients to connect before starting the game. Player id's are given in the same order as the connection order Clients are launched with:

```
java -cp ./path/to/projectsrc.jar boardgame.Client [Class [serverName [serverPort]]]
```

where:

- `Class` is the plauer to be run (default=halma.CCHumanPlayer)
- `serverName` is the server address (default=localhost)
- `serverPort` is the port number (default=8123)

For example:

```
java -cp ./path/to/projectsrc.jar boardgame.Client halma.CCRandomPlayer localhost 8123
```

launches the random Halma player client, connecting to a server on the local machine and default TCP port. The game starts immediately once two clients are connected.

If using the server GUI, it is also possible to launch clients to be run on the same machine as the server from the "Launch" menu. This starts a regular client running in a background thread.

4 Launching the Log File Viewer

It is possible to run the server GUI without starting a server to examine existing log files. Moves are loaded from the file as though the game was being played on a server.

```
java -cp ./path/to/projectsrc.jar boardgame.ServerGUI [filename]
```

where `filename` is an optional log file to open.

As an example:

```
java -cp ./path/to/projectsrc.jar boardgame.ServerGUI logs/game00001.log
```

allows you to review the complete sequence of moves in your first game.

5 Human Players

5.1 On the Server GUI

To play against a software player or another person, launch the server with a GUI. The status bar indicates that the server is waiting for a client connection. Select “Launch human player” from the “Launch” menu to start a user client which will obtain moves through the GUI. Launch another client from the command line or from the “Launch” menu. You can make moves for the human player by dragging a token to a desired location.

5.2 Remote Human Client

Humans can also connect to a remote server by passing the Player class `halma.CCHumanPlayer` to the client software. This player obtains moves from a GUI on its turn. For example:

```
java -cp ./path/to/projectsrc.jar boardgame.Client halma.CCHumanPlayer
```

brings up a GUI to choose moves for the client.

Set a large time-out (at least a few minutes) for the server to support remote human clients since the server doesn’t know whether it is communicating with a computer or human player.

6 Implementing a Player

To implement a new player, extend the `boardgame.Player` class, and pass the new class as the argument to the client software. See the file **Player.java** for further instructions, and **submission_example** folder for an example.

To launch custom players from the GUI, add the class name to the array `ServerGUI.PLAYER_CLASSES` in the **ServerGUI.java** file.

Please read carefully the project specifications on how to structure your code!

6.1 Classpath

You can make your player visible to the provided code by including it in the classpath by replacing every occurance of `-cp ./path/to/projectsrc.jar` with:

```
-cp ./path/to/projectsrc.jar:/path/to/your/class/files/sXXXXXXXXX
```

6.1.1 Windows users

Please note that the classpath syntax is slightly different for windows command line. Instead of the separator ‘:’, use ‘;’. Also make sure the path syntax itself is in windows format.

6.2 Eclipse users

You can link to the jar using

```
Build Path --> add External Archives
```

or alternatively, you can import the whole jar files. Please make sure your submission format is correct before submitting.