Hus Project Package

Eric Crawford (eric.crawford@mail.mcgill.ca)

February 4, 2016

1 Overview

An implementation of the Hus game is provided. To play a game, a server (the board) and two clients (the agents) are launched, and the clients connect to the server via TCP sockets. This allows the clients to be run on separate computers. A GUI is provided, which can be used to display an ongoing game or to examine log files of previous games. Games between software agents can be played without the GUI for increased speed.

The provided source code for is located in the **src** directory. You will create your agent by editing the code in the **student_player** package. You should not edit the **boardgame** or **hus** packages. The source code is structured as follows:

```
src
|--- boardgame                    A package for implementing board games. Provides
|                                 infrastructure for logging, the GUI, and communicating
|                                 via TCP. You can largely ignore this package.
|
|--- hus                          An implementation of Hus.
|  |-- HusBoardState.java         All game logic can be found here. Also provides methods
|  |                              that agents can use to interact with the game.
|  |
|  |-- HusPlayer.java             An abstract class for all Hus players. You will provide
|  |                              an extension of this class.
|  |
|  |-- RandomHusPlayer.java       An example extension of HusPlayer that plays randomly.
|  |
|  |-- HusMove.java               A Hus move.
|  |
|  |-- HusBoard.java              Maintains an instance of HusBoardState, and is used
|  |                              by the server and GUI to interact with the board state.
|  |
|  \-- HusBoardPanel.java         Displays a Hus game in the GUI.
|
|--- student_player               A package containing your agent.
|  |-- StudentPlayer.java         The class that you need to implement.
|  |
|  \--- mytools                   A placeholder package for extra code that you write.
|     \-- MyTools.java
|
\--- autoplay                     A script for running large batches of games.
   \-- Autoplay.java
```

2 Workflows

You are free to develop your agent using whichever Java-based workflow you choose. However, we have provided specific support for two of these workflows. If using a different workflow, you will need a means of compiling the source code.

2.1 Ant

Ant is to Java as **make** is to C. We have provided a build.xml file (the Ant equivalent of a Makefile) in the root of the project package. This file provides short commands for common tasks such as compiling the provided code and launching servers and clients. If you have Ant installed, the commands can be run from the command-line using the syntax: **ant** ⟨**command**⟩. The provided commands are:

- **clean** – Delete the **bin** directory.
- **compile** – Compile the code in release mode. Stores the class files in **bin**.
- **debug** – Compile the code in debug mode. Stores the class files in **bin**.
- **student** – Launch a client with the student player.
- **gui** – Launch the Server with the GUI enabled.
- **server** – Launch the Server with no GUI, and the -k switch provided so the server restarts after each game.
- **autoplay** – Run the Autoplay script. By default, runs 2 games. To run a different number of games, supply an argument like **-Dn_games=10** to the ant command.

Feel free to modify build.xml to suit your needs.

2.2 Eclipse

The root directory of the project package is a valid Eclipse project. Import it into Eclipse using **Import** → **General** → **Existing Projects into Workspace**, and then select the root of the project package. Also, the **eclipse** directory contains a number of launch configuration files; these can be imported into the project by right-clicking the Eclipse project, selecting **Import** → **Run/Debug** → **Launch Configurations**, and then navigating to the **eclipse** directory. These run configurations can be further customized as needed. Note that the default behaviour of Eclipse is to automatically compile your code as soon as you save it and store the class files inside the **bin** directory, which is exactly what we want.

3 Playing Games

Here we provide documentation for the server and client programs. The commands outlined in this section are the commands that are called by the provided build.xml file (if using ant) and launch configurations (if using Eclipse). The details provided in this section are especially useful for advanced use of the provided code, such as playing games where the clients are located on different machines than the server. All of these commands assume that you have compiled the code and stored the class files in a directory named **bin** located inside the root directory of the project package

3.1 Launching the server

To start the server from the root folder of the project package, run the command:

```
java -cp bin boardgame.Server [-p port] [-ng] [-q] [-t n] [-ft n] [-k]
```

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=2000)
- (-ft n) sets the first move timeout to n milliseconds. (default=60000)
- (-k) launch a new server every time a game ends (used to run multiple games without the GUI)

For example, assuming the current directory is the root directory of the project package, the command:

```
java -cp bin boardgame.Server -p 8123 -t 300000
```

launches a server on port 8123 (the default TCP port) with the GUI displayed and a timeout of 300 seconds.

The server waits for two clients to connect. Closing the GUI window will not terminate the server; the server exits once the game is finished. If the **-k** flag was passed, then a new server starts up and waits for connections as soon as the previous one exits. Log files for each game are automatically written to the **logs** subdirectory. The log file for a game contains a list of all moves, names of the two players that participated, and other parameters. The server also maintains a file, **outcomes.txt**, which stores a summary of all 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.

3.2 Launching a client

As stated previously, the server waits for two client players to connect before starting the game. If using the GUI, one can launch clients (which will run on the same machine as the server) from the **Launch** menu. This starts a regular client running in a background thread, which plays using the selected player class. In order to play a game of Hus using the GUI by clicking on pits to select moves, choose **Launch human player** from the **Launch** menu.

Clients can also be launched from the command line. From the root directory of the project package, run the command:

```
java -cp bin boardgame.Client [playerClass [serverName [serverPort]]]
```

where:

- playerClass is the player to be run (default=hus.RandomHusPlayer)
- serverName is the server address (default=localhost)
- serverPort is the port number (default=8123)

For example, the command:

```
java -cp bin boardgame.Client hus.RandomHusPlayer localhost 8123
```

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

The two approaches of launching players from the GUI and launching players from the command line can also be combined. For instance, one can use the GUI to manually play against the random player (or any other agent) by first launching a human player using the GUI, and subsequently launching the random player, either by selecting it in the GUI or running the appropriate command from the command line. The order can also be switched; the player that connects to the server first plays as player 0 and moves first.

3.3 Autoplay

The provided Autoplay script can be used to play a large batch of games between two agents automatically. It launches a Server with the options **-k -ng**, and then repeatedly launches pairs of agents to play against one another. To use Autoplay, run the following command from the root directory of the project package:

```
java -cp bin autoplay.Autoplay n_games
```

where **n_games** is a positive integer number of games to play. The default behaviour of Autoplay is to play the student agent against the random player, with the random player agent going first every second game. You can modify this behaviour by editing **Autoplay.java**.

4 Implementing a Player

New agents are created by extending the class **hus.HusPlayer**. The skeleton for a new agent is provided by the class **student_player.StudentPlayer**, and you should proceed by directly modifying that file. In implementing your agent, you have two primary responsibilities:

1. Change the constructor of the **StudentPlayer** class so that it returns your student number.
2. Change the code in the method **chooseMove** to implement your agent's strategy for choosing moves.

To launch your agent from the command line, run the command:

```
java -cp bin boardgame.Client student_player.StudentPlayer
```

You can also launch your agent by selecting it from the **Launch** menu in the GUI. **The project specification has further details on implementing your player, and, in particular, what you need to submit.**