CSE 571 Project Description (GGP Track)

Instructions:

Setup:

Note: you must have java installed on your machine to invoke the game controller and the game interface. The latest version of the java JDK/JRE can be downloaded and installed from http://www.oracle.com/technetwork/java/javase/downloads/index.html
(the java JDK includes the java JRE but only the JRE is required since this project does not require any modifications to the java code)

Once you have downloaded the archive for the project corresponding to your operating system, decompress it to a directory of your choice.

Game Controller:

Open a terminal or command line and navigate to the directory to which you extracted the project files. Then use the following command to open the Game Controller:

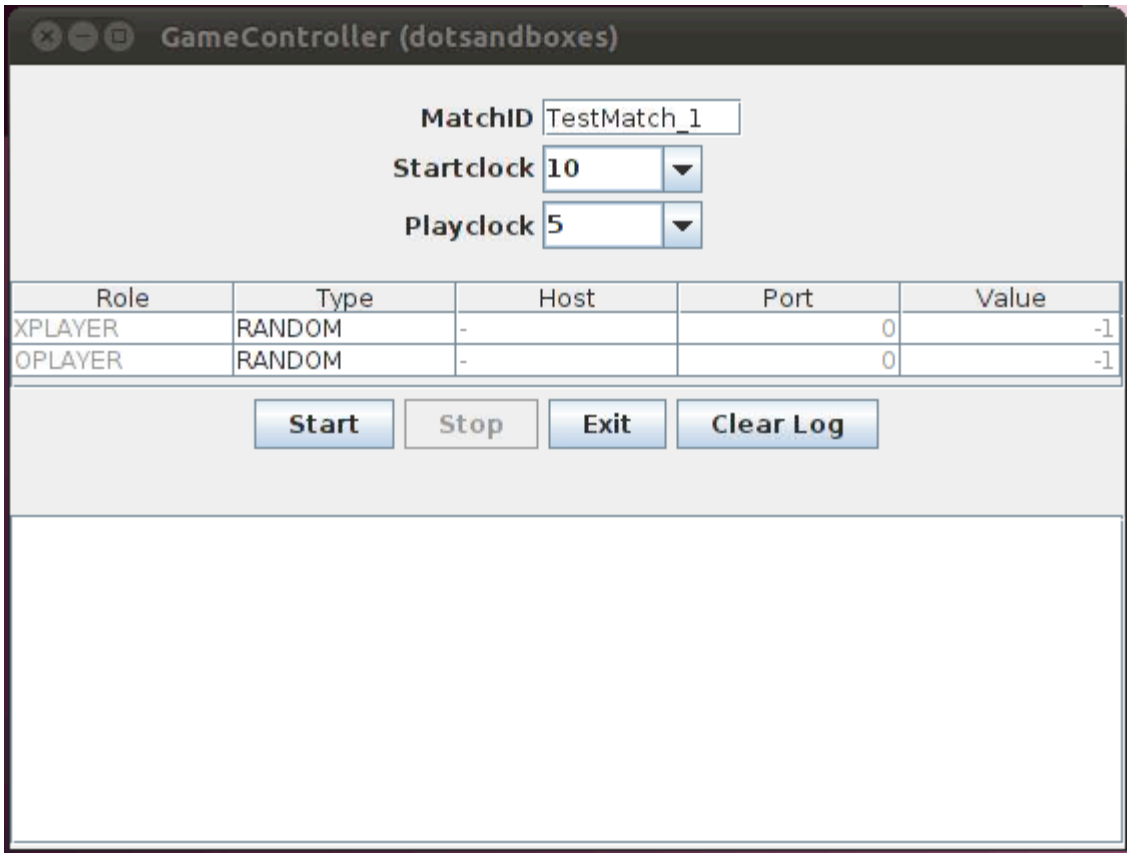
./gamecontroller-gui-r466.jar & (linux)
game gamecontroller-gui-r466.jar (windows)

Note: The permissions of the files should allow you to run these without issue but in case you do get a permission denied, use the chmod command to change the permissions e.g. “sudo chmod 777 <yourfile>” to make <yourfile> readable, writable, and executable for all users.

Now a dialogue box asking for the game description should appear. Navigate to directory to which you extracted the project files and select the description (e.g. DotsAndBoxes.gdl) and press Start

Game.

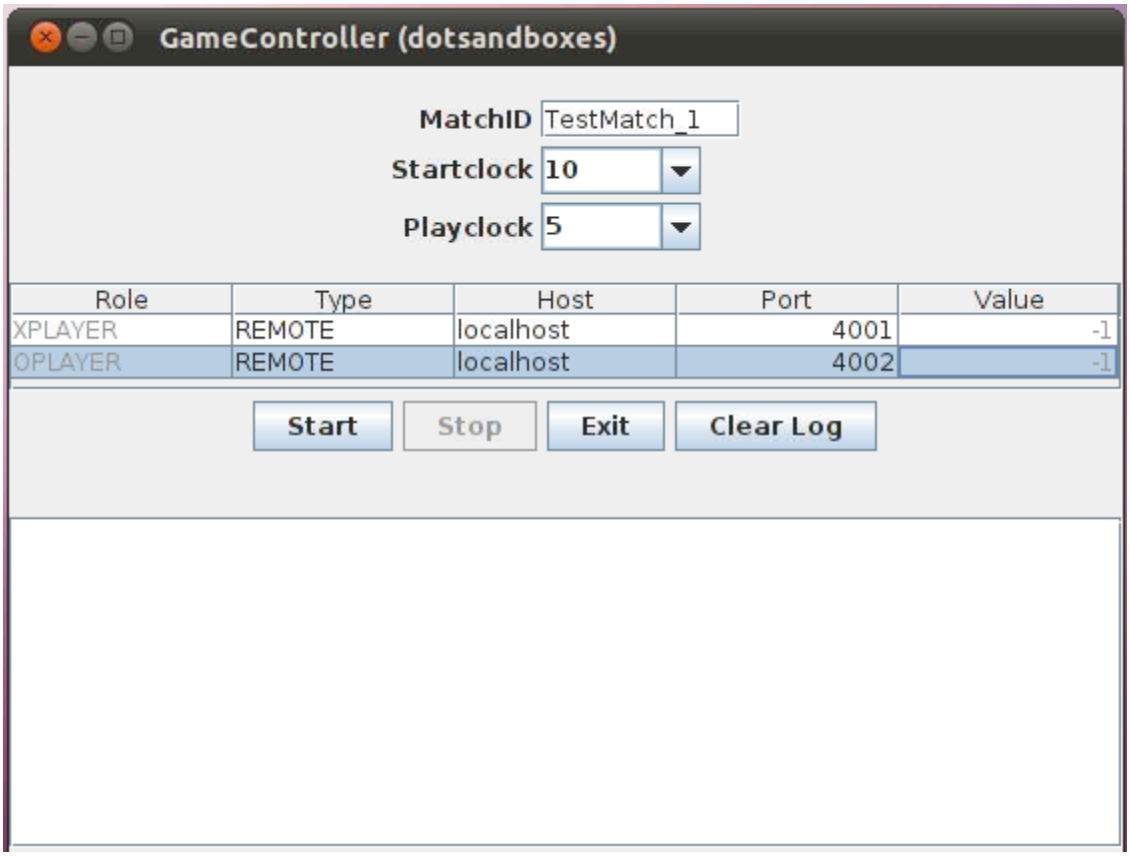
Now the controller should appear, which will allow you to specify the configuration of the players:



The Start clock specifies the amount of time (in seconds) from sending the description of the game to your player until the game will start (this is a general game playing controller so this time would be used by the players to develop their strategies; however we will already have the game ahead of time and the strategies developed so this is not very applicable).

The Play clock specifies the amount of time (in seconds) that the player has to send its move or a random legal move.

The two rows are the two players who will participate in the game. The type field has 3 options: Random, Remote, and Legal. Remote is what you can use to test your player. In this case, you will also be asked to specify a port. You can test your player against a random player (don't worry, the moves will be legal, it can't cheat) or another remote player using the Random or Remote options respectively (you should not need to use the Legal option; it is not very helpful either as it simply systematically chooses the "first" available legal move). This can help to see how your new idea compares to your old one. However if you specify 2 remote players, they must be on different ports. Here is one example configuration:



Before pressing start, your players need to be running.

Game Interface Files

For each remote player (every file listed here will have a “2” in place of the “1” for the second player if you run 2 players) you will be running, there are several files in your local directory:

dots-<your-firstname-in-low-case>.lp (e.g., dots-joohyung.lp) This is your game AI that determines the next move and the vast majority of where you will be working (you may also want to do some testing with the dots-state1.lp to ensure your strategy is being followed as expected). The interface expects the output of this logic program to be an answer set with a single atom “move(X1,Y1,X2,Y2)”

dots-state1.lp This is a record of the current state of the game (it doesn’t matter what you put in here when running with the game controller, the game interface will overwrite it). The state of the game for dots and boxes is simply a collection of the lines that have already been drawn expressed as “line(1,1,1,2).” Which indicates that a line exists between (1,1) and (1,2).

dots1.sh (linux) or
dots1.bat (windows)
This is the script that will run your game AI with the current state and output it to a file move1.text. As this is the output of your game AI, the format of move1.txt will be assumed to contain a single “move(X1,Y1,X2,Y2)” on the second line of the file. The shell script (linux) is simply one line

```
f2lp dots-state1.lp dots-$1.lp | gringo | claspD > move1.txt
```

The batch script (windows) is just two lines:

```
f2lp.exe dots-state1.lp dots-%1.lp | gringo | claspD > move1.txt
EXIT
```

The files for omok are similar and the differences in the input (omok-state1.lp) and output (output of claspD on omok-state1.lp and omok-<name>.lp) are described in the input/output section. The difference with the omok scripts is that the constant color will be specified (black for player 1, white for player 2 according to the game interface command line order). The shell script is one line:

```
f2lp omok-state1.lp omok-$1.lp | gringo –c color=black | claspD > move1.txt
```

The batch script is just two lines:

```
f2lp.exe omok-state1.lp omok-%1.lp | gringo –c color=black | claspD > move1.txt
EXIT
```

Running the Game Interface

Now you can run the GameInterface.jar file but you must specify on the command line the port number (these must coincide with the port numbers specified in the game controller) and game name for each player. Examples below for two cases:

GameInterface.jar “4001” “dots” <name1> “4002” “dots” <name2> (this will create 2 players, name1 and name 2, and 2 connections on a windows system for the dots and boxes game)

./GameInterface.jar “4001” “omok” <name1> (this will create 1 player, <name1>, and connection on a linux system for the omok game)

Now press start game in the game controller and see how your player does

Warning: if you stop the game via the game controller and start it again, the game interface (GameInterface.jar) will be confused and not interpret the messages correctly so if you must stop the game, you should also restart the interface.

Disclaimer: The interface is a work in progress so the version used in the final may use different naming conventions or have different visualizations but renaming files aside, this should not affect the compatibility with your game AI asp files (the only files you will need to submit).

Input/Output

ASP Input / Output: For Dots and Boxes, the dots-state1.lp will contain a set of facts corresponding to the lines that have already been drawn. These facts are of the form “line(x1,y1,x2,y2).” The expected output from the dots-<name>.lp file is an answer set with a single atom “move(x1,y1,x2,y2)” shown. To hide all but this, include the following lines in your asp file:
#hide.
#show move/4.

Sample input (dots-state1.lp)

```
line(1,1,1,2).
line(1,1,2,1).
line(1,2,1,3).
```

Sample output (results of ASP call stored in move1.txt)

```
Answer: 1
move(6,5,6,6)
SATISFIABLE
Models       : 1+
Time         : 0.000
Prepare    : 0.000
Prepro.    : 0.000
Solving    : 0.000
```

For Omok, the omokstate1.lp will contain a set of facts corresponding to the pieces that have already been laid on the board. These facts are of the form “piece(color,x1,y1).” indicating a piece of color “color” is at position (x1,y1). Note that your omok-<name>.lp file should have a constant “color” that will be specified by the script files by the commandline option “-c color=black” or “-c color=white”. Your

strategy will use this information to determine which color it is representing and thus, what a “good” move would be. The expected output from the omok.lp file is an answer set with a single atom “move(x1,y1)” shown. To hide all but this, include the following lines in your asp file:
#hide.
#show move/2.

Sample input (omok-state1.lp)

```
piece(black,5,5).
piece(white,5,6).
piece(black,6,5).
```

Sample output (results of ASP call stored in move1.txt)

```
Answer: 1
move(7,5)
SATISFIABLE
Models       : 1+
Time         : 0.000
Prepare    : 0.000
Prepro.    : 0.000
Solving    : 0.000
```