CS221 Project03 – Mancala tournament

Paul Haskell

INTRODUCTION

In Project02, you programmed your computer to play Mancala against a human player. Recently, we learned how to have our programs communicate with other programs via TCP/IP networking. So obviously, we gotta have our Mancala programs play against each other! That's what we'll do in **Project03**.

If you want to modify your playing strategy from what you had before, that's 100% fine.

Your program

Your program should be called **mancala3.c** . Your program is not required to print out Mancala boards or prompt human players…your program will play against other students' Mancala programs. (You can print out whatever information you desire, but no printing is required.)

Networking

This project must be done in the **vlab** environment, so all our programs can connect with each other. Your program will connect to a central server, which will relay your game moves to another player. Your program will play several games per second, against a different opponent each time.

Your program must `connect()` to the server, whose IP address and port you should read from the command line.

Communication Protocol

The server will send your program a sequence of commands, which your program shall read again and again. Some commands require that your program write back a reply. Every message you read and write has 1 byte at the start which gives the length of the following message. The byte value is a raw unsigned char number, not an ASCII value.[1]

Here is a list of the server commands and required responses:

- **LOGIN**
    - The length before the "LOGIN" string is 6, of course.
    - You must reply with a string containing your GithubUserID, a ':', and an Avatar name, e.g.
      `BrockPurdy:QB1`
    - Don't forget to prepend the length of the message before the actual data! And don't forget the \0 at the end

[1] For your info, Java's read and write methods handle the `String` lengths automatically for us.

- **NEWGAME**
    - No reply is needed. This tells your program you are about to start a new game.
- **PLAY**
    - You must reply with a single ASCII digit between '1' and '6'. This is your game move.
    - Don't forget to prepend a length of 2 to your reply
- **OPP:<<some number>>**
    - No reply is needed
    - The "some number" is a single ASCII digit between '1' and '6'. This is your opponent's move.
- **DONE:<<some message>>**
    - No reply is needed
    - The server sends this message to your program when the server thinks your program did something wrong: waiting too long to reply to a command, sending an invalid response to "LOGIN", etc. Please print out the message to see what happened.

Your program must keep track of your moves and your opponent's moves to maintain the proper state of the Mancala board. (The server does this also.) All of you plays must be legal—if you play an empty square, the server will know and will kick you out of the game.

Some hints

Your program should have some sort of reading method that reads the message length, reads the actual message, handles errors, and returns the message. You probably want a similar method for writing.

The `read()` and `write()` system calls are very slow. To speed up your program, your writing method must call `write()` only once per message, to write both the message length and content.

When you read, there is a danger of reading two back-to-back messages from the server in a single `read()` command. You either can handle that via some fancy programming (write a "buffered reader"), or just read each message in 2 steps:

- Read a single byte, to get the length of the actual message
- Then read the actual message payload

CONCLUSION:

Please push your program to GitHub before the project deadline. The deadline for completion of Project03 is 11:59pm Wednesday May 7.

Task	Score, points
Mancala3.c compiles, runs, and connects to the server	5
Mancala3.c follows the rules of the game, runs games to their proper conclusion, and starts new games	15

Mancala3.c uses an intelligent and effective playing strategy, as judged subjectively by the grader	16
Software and design quality, as judged by the grader	20