User Manual

HeroChess Version 2.0

May 17, 2021

Paul John Lee, Irania Ruby Mazariegos,
Rachel Villamor, Keane Wong
EECS 22L, University of California Irvine, Irvine, CA 92697

Contents

Glossary of Networking Terms

Client: any hardware or software that connects, uses, and/or communicates with a server
Connection: session by which information is transmitted and received throughout a network
Host: any hardware that permits service and communication to other hardware or networks
IP address: a unique string of characters that identifies each computer using the Internet Protocol to communicate over a network
Network: two or more computers connected together for electronic communication
Port: virtual endpoint that is associated with a host's IP address and serves as a means of communication between a server and application
Protocol: a set of rules that describe the how data is communicated and processed between devices within a network
Server: a computer or computer program which manages access to a centralized resource or service in a network.
Session: a temporary period of communication between hardware or users
Socket: A term used to refer to a single end of a communication channel used to manage data transfer
Error Messages: Check the "Error Messages" section for any networking error during the game.

1 Installation

1.1 System requirements

To use this version of the HeroChess Multiplayer functionality, the user(s) must have access to the EECS Linux Servers, or a suitable host with a stable release of the program. Users are recommended to have the Linux version CentOS 6.10 and perhaps even Xming if the graphical user interface is included. The amount of memory required for the program to run is still to be determined. All users that use the multiplayer functionality must have access to an internet connection for the duration of their playtime.

1.2 Setup and Configuration

Download a SSH client with terminal support.

1. On the terminal type 'tar -xvzf `P2_V2.0`.tar.gz' and press enter.
2. Then, type 'cd `P2_V2.0`' and press enter.
3. Finally, type 'make' and press enter. Installation is now complete.
4. To run the game, type './bin/chess', press enter, and enjoy the game.
5. Type and enter './client 11900'
6. For those running the server end of this program, follow the same steps up until number 4, where instead you type and enter './server 11900'

1.3 Uninstalling

Users will be able to uninstall the Chess game executable files by being in the directory called `P2_V2.0` and using the 'make clean' command.

2 Chess V2 - Client

2.1 Overview of Features

ChessV2 is the extended version of Chess V1.0. It will support client system and multiplayer features on the UCI servers along with all these features from the previous version. Players can create an account with a custom username and password which will remain encrypted and confidential permanently. Account will contain individual information in the game, such as player name, win/losses ratio, and the player rating (ELO).

2.2 Logging In/Registering

When the client first opens ChessV2, they will see a menu asking if they are a new user or a returning user: Please select 1 or 2

1. New user
2. Returning user

New clients will be prompted to create an account by inputting a unique username and password. The username must be between 6-8 characters. If the username entered exceeds the character limit or is already taken, a message will be shown asking for a different username. The password must be 8 characters long, contain one number (0-9), and one special character ($, @, !, *). The registering process will be the following:

Enter a unique username (6-8 characters): petertheanteater
Username too long!
Enter a unique username (6-8 characters): peterant

Once the account is created, the client will be taken to the main menu of ChessV2. Returning clients will be prompted to enter their username and password to log in to their account. The log in process will be the following:

Username:
Password:

If the username entered does not match a registered user, the client will be prompted to create an account:

Username:
Username does not match an existing account! Enter '1' to register for an account...

If the password entered is incorrect, the client will be given the option to try again or change their password.
Password:
Password is incorrect! Try again or change your password...
Try again
Change password

If the client decides to change their password, the same process used to register for an account will be used. Once the client enters the correct information to access their account, they will be taken to the main menu of ChessV2.

2.3 Getting Started

Players will first see the login screen where they either create an account or login to the existing account. After logging in, players will see the starting screen with Human vs. Human, Human vs. AI, game settings, account settings and exit game.

To play a game in the multiplayer game mode, players must have access to the other player through the UCI server. After entering the ip address of the enemy player, the

player will be connected and the game will print a chess board with the standard initial condition for the user to play with.

Possible feature: game will detect all the open rooms for a chess game on the server and list them into a specific format.

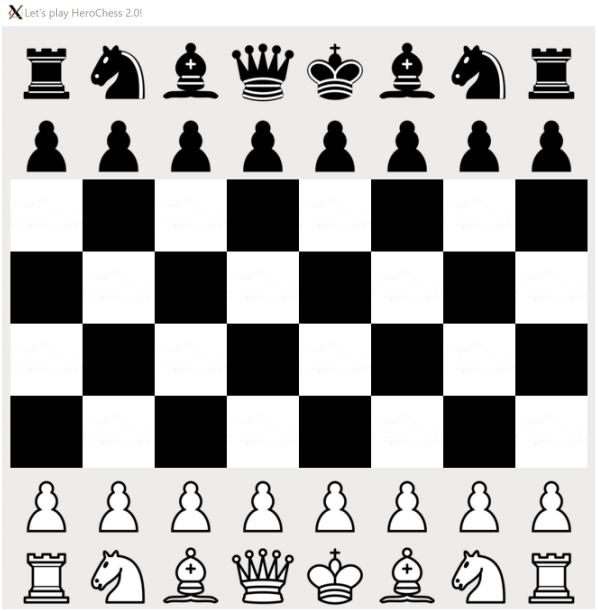
2.4 Usage Scenario (Playing a Game - Client View for Both Players

Upon logging in as a user, both clients will see the main menu to the chess game. Here, they will have the option to start a game, change game settings (colors, etc.), or go over the rules of chess.

When starting a game, the users will see a basic starting chess board before them. The right side will show the y-axis of the board with labels of numbers 1 to 8. The bottom of the screen will show the x-axis with labels of letters 'a' to 'h'. At the start of the game, the users can decipher which group of pieces is theirs by looking at the left side of their screen where there are name labels positioned at the top and bottom of the board. The position of the labels corresponds to the initial position of the user's pieces. For example, if the name label "You" is at the bottom left of the screen and the current blank board has all of the white pieces positioned at the bottom, then the current player will have control over the white pieces throughout the game. The other name displayed will be the username of the other player.

Throughout the game, prompts will appear on the screen asking the player to enter the positions of pieces they want to move and the positions of the locations they want the piece to move to. Similar to HeroChess v1.0, player inputs will be taken as keyboard inputs. Inputs must follow the format of letter-number, e.g. a4 or A4.

Fig. 1: Example of Initial GUI Board Setup



3 Chess V2 - Server

3.1 Overview of Features

For this project, in addition to adding features that allow users to better interact with one another, this project will include a server provider in the form of a simple active program that listens for and responds to client requests and stores relevant data necessary for the code's functionality. Relevant data includes information such as account information, and associated verification data such as passwords, usernames, IP addresses, and port numbers. All of these will be used in conjunction with one another to collectively create a cohesive network of user identities, all of which will be generally handled server side. Data stored is kept in a combination of binary data and persistent text file data that is accessible through privately handled functions that accesses and displays it as needed without any unnecessary input from the user.

The user side of the program requires a chat functionality that, besides needing to ensure the correct users are talking to one another, that their messages are being actively listened for in a timely manner, and like most chat user interfaces, accurately displays the correct users in chat and allows for challenges between friendly users.. The user side allows for data updates as necessary, and of course the server side supports any changes accurately.

All of this will be handled in some way through a program executable hosted on a separate machine that, over the internet, connects multiple players while each player, once installed the correct files, can play online.

3.2 Usage scenario (general, starting server)

The program itself has an executable command that can be run like any program, and runs off the local machine to become a hosting server. Once booted up, the server will begin creating a number of sockets in which users can communicate with it, and binding them to ports on the machine, each of which then go on to listen to requests and communicate data. This is what listens for client side instigation as the program then goes on to verify users or register them, then play games amongst one another. A sample log of a server usage is below, detailing startup:

Fig. 2: Sample server usage log

```
Server: STARTING_UP
Server: .. . . . . . ..
Server: SOCKETS CREATED

Server: SOCKET BOUND TO PORT ADDRESS 11900
Server: SOCKET BOUND TO PORT ADDRESS 11901
.. .. .. .. ..
(Server initializes ports as necessary and to useful capacity)

Server: INITIALIZED SOCKETS NOW LISTENING
Server: .. . . . . . ..

Client: CONNECTING_TO_SERVER
Server: OK ACCEPTING REQUEST

Server: Requesting USER_NAME
```

3.3 Logging in/registering (however you support uniquely identifying clients)

Upon connecting to the server successfully, the program will prompt the user to either log in or sign up, at which point the server will then be actively engaging with the user through an established port. From there, the server will have then opened a socket to respond to and process data with. The inputted user data will then be run through a database of established users, or appended to it, as appropriate, and then further checked against their associated password, kept in the same file and synchronously updated.

From the server side, any administrators managing the provider will have access to the ports and open sockets that stay open to queries sent from potential players. Any changes to it are reflected by messages that mark client-server communication, as well as errors. For example, a typical server communication is detailed below:

Fig. 3: Typical Server Communication

```
(Selecting the option for new user, option 1 on the user side)
Client: 1
Server: OK Requesting USER_NAME

Client: USER_NAME myUser2021
Server: ERROR CODE 100 Username already taken

Client: USER_NAME myUser 2022
Server: OK Requesting Pass_Word

ClientL PASS_WORD MyPassWord
Server: OK Registration Successful

(Client now selecting option for logging in, using their newly
registered username)
Server: OK Requesting User_name
Client: USER_NAME myUser2022

Server: OK Requesting Pass_Word
Client: PASS_WORD MyPassWord

Server: OK Confirmed
Server: OK Logged in
```

3.4 Detailed description of chess game integration

The Chess Game runs primarily on the server but has checks as well as client side. Some of the appropriate game checks, especially for legality and simple checks like bounds, run on the client side to help reduce runtime and stress on the server end. However, the game is ultimately run on the server to ensure a synchronous multiplayer experience. On the client ends of the program, there are minimal running processes, mostly focusing on taking in valid inputs and sending them to the server. On the server end of it, the program looks to be playing an ordinary game of chess, much like with the single player mode, but the moves are fed in directly to the code as pre-processed data from two different machines. The example below demonstrates a typical view of the program from server side:

Fig. 4: Typical Server View of the Program

```
    Client: REQUESTING_GAME_BOARD
    Server: OK |__|__|__|__|........

    Server: REQUESTING_MOVE_CHOICE
    Client: SOURCE_SQUARE E2 TO E4
    Server: OK confirmed

    (Server processes move as usual, checking for checks,
checkmates, legality, and logs move into internal data
structure)

    Server: MOVE_VALIDATED
    Client: OK Confirmed

    Client: REQUESTING_GAME_BOARD
    ServerL OK |__|__|__|__|........
```

Copyright

Error Messages

The error messages of this program are labeled by a 3 digit error code, each digit denoting a different aspect of the error. The first digit denotes where the error occurred, the second digit typically corresponds to a class of errors that occurred in that location, and the last digit denotes the specific error in that class. Below are detailed some common denominations for this error labeling.

List of common 4XX errors
400 Bad Request - The server could not understand the command due to invalid syntax.
401 Unauthorized - the current user lacks valid authentication credentials for the target resource.
403 Forbidden - The clients does not have access rights to the content
404 Not Found - The server cannot find the requested resource. Links that lead to a 404 page are often called broken or dead links and can be subject to link rot.
408 Request Timeout - This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection.
List of common 5XX errors
502 Bad Gateway - This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.
503 Service Unavailable - The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded.
504 Gateway Timeout - The server is acting as a gateway but cannot get a response in time.
511 Network Authentication Required - This status code indicates that the client needs to authenticate to gain network access.
In General, the first digit 4 corresponds to a client side error, meaning some issue from the user end, and the first digit 5 corresponds to a server side error, meaning some issue from the administrative end.
Other status codes include 1XX, 2XX, and 3XX, however these are typically not relevant to errors and are not supposed to be seen by users.

Index