MULTIPLAYER CHESS

Software Specification

Version 1.1

By:

Electric Lizards

UC Irvine

Bokor, Jacob	jbokor@uci.edu
Lam, Rayi	rayil@uci.edu
Nghi Nguyen	nghihn2@uci.edu
Shum, Ryan	rwshum@uci.edu
Wang,Lichen	lichew1@uci.edu

Development Glossary

AI: Artificial intelligence; an in-game entity that uses computer code to make decisions.

Bug: An error in a computer program.

Datatype: An attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

Struct: In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.

String: An array of characters as a variable.

Pointer: a variable whose value is the address of another variable; a link.

Socket: A socket is one endpoint of a two-way communication link between two programs running on the network.

Host: A host (also known as "network host") is a computer or other device that communicates with other hosts on a network. Hosts on a network include clients and servers -- that send or receive data, services or applications.

1. Software Architecture Overview

1.1. Main Data Types and Structures

- Board - 2D string array
- Username - char pointer
    - Usernames are stored in a text file for the server to read from
- User's password - char pointer
    - Passwords are stored in a text file for the server to read from
- Message - char pointer

1.2. Major Software Components

- main.c
- server.c
- client.c
- account.c
- board.h, board.c

Refer to the Chess version 1.0 software specifications for the remaining software components.

1.3. Module Interfaces

1.3.1. main.c

1.3.1.1. server

1.3.2. account.c

1.3.2.1. Handle accounts

1.3.3. gameClient.c

1.3.3.1. Client

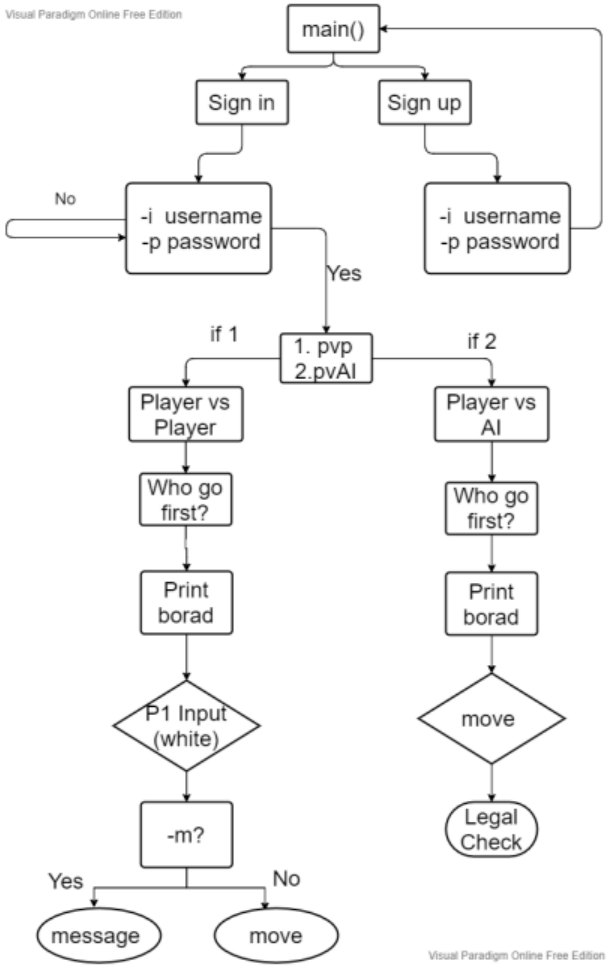
1.3.4. server.c

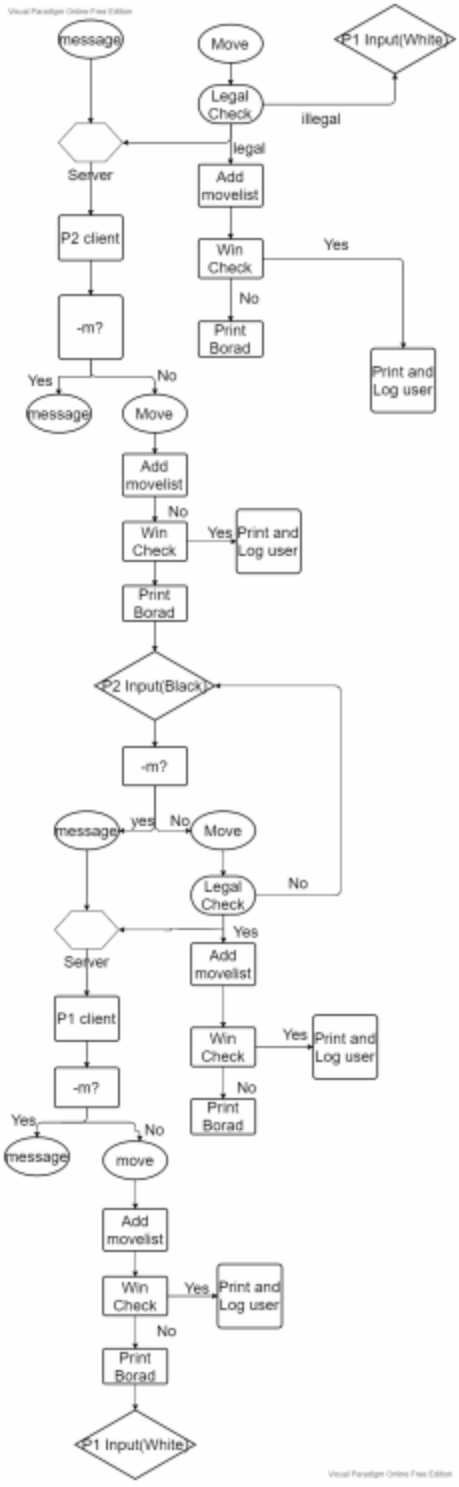
1.3.4.1. Server initialization

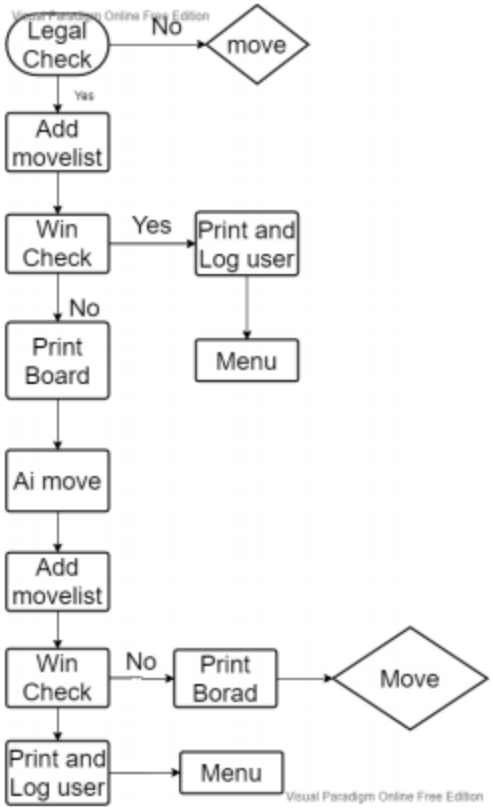
Refer to the Chess version 1.0 software specifications for the remaining module interfaces.

1.4. Overall Program Control Flow

1. Display login menu for users to log in
    a. Login
    b. Exit
2. Print the board
3. White player inputs move or message
4. Check for input error
5. Check if white side has a check or checkmate
6. Print the board
7. Black player inputs move or message
8. Check for input error
9. Check if black side has a check or checkmate
10. Repeat steps 4 through 11
11. Print out the winner
12. Save chess game replay to a text file
13. Exit




message
Move
P1 Input(White)
Legal Check
illegal
legal
Server
Add movelist
P2 client
Win Check
Yes
No
Print Borad
-m?
Print and Log user
Yes
No
message
Move
Add movelist
No
Win Check
Yes
Print and Log user
Print Borad
P2 Input(Black)
-m?
message
yes
No
Move
Legal Check
No
Yes
Server
Add movelist
P1 client
Win Check
Yes
Print and Log user
-m?
No
Print Borad
Yes
No
message
move
Add movelist
Win Check
Yes
Print and Log user
No
Print Borad
P1 Input(White)


Legal Check
No
move
Yes
Add movelist
Win Check
Yes
Print and Log user
No
Print Board
Menu
Ai move
Add movelist
Win Check
No
Print Borad
Move
Print and Log user
Menu

2. Installation

2.1. System Requirements, Compatibility

2.1.1. System Requirements

Operating System: Linux
Storage Requirements: 10mb

2.1.2. Compatibility

Linux EECS Server

2.2. Setup and Configuration

The entirety of the program will be within a tarball file (.tar.gz). Once extracted, the program will be in a folder called ./chessgame/ located at the location where the tarball was extracted. Navigate to the root directory of the program and proceed to the following step.

2.3. Building, Compilation, Installation

Once in the root directory, running the “make” command will compile and combine the modules files into one executable file. The makefile will compile the program in compliance with the ANSI C99 Standard for all of the modules.

After compiling, the generated executable can be run with command “ ./chessgame “.

3. Documentation of Packages, Modules and Interfaces

3.1. Detailed description of data structures

3.1.1. Critical snippets of source code

For each account that is created, the account will hold the userID, the username, password, socket number, opponent's socket number, and a list of the user's friends. The existing accounts will be held in a linked list.

```
typedef struct accountEntry {
        int userID;
        char user[20];  // username
        char pass[20];  // password
        int socket;  // 0 if not connected/online
        int opponentSocket; // 0 if not playing against an opponent
        int friends[10];  // storing another user's userID
        account* next;
        account* prev;
} account;

typedef struct accountList {
        account* first;
        account* last;
        int length;
} accountList;
```

For remaining data structures, refer to the chess version 1.0 software specifications.

3.2. Detailed description of functions and parameters

Client Side Functionality

- gameClient.c
    - Int main()
        - Parameter: none
        - Return: 0
        - Process: The main will start the client, send and receive message from the server
    - int printMenu(char *data)
        - Parameter: None
        - Return: an integer

        - Process: Prints the menu and take in the input of the client to either login or exit

Server Side Functionality

- server.h, server.c
    - Int serverInit(short port, int backlog)
        - Parameter: port number, and backlog
        - Return: server socket
        - Process: initialize server, setting up
    - int acceptConnection(int serverSocket)
        - Parameter: server socket number
        - Return: client socket
        - Process: accepts all connection of the server
    - int doStuff(int clientSocket, accountList *list)
        - Parameter: client socket number and the list of accounts
        - Return: 0
        - Process: Handle parsing of data sent and sends data to other client
- main.c
    - int main()
        - Parameter: None
        - Return: 0
        - Process: The main will start the server and handle the client inputs and returns them.
- account.h, account.c
    - accountList *createAccountList()
        - Parameter:none
        - Return: a new account linked list
        - Process: create new account linked list
    - void deleteAccountList(accountList *list)
        - Parameter: account list
        - Return: none
        - Process: delete an account list
    - account *createAccount()
        - Parameter: none
        - Return: a new account
        - Process: Create a new account linked list
    - void deleteAccount(account *acct)
        - Parameter: account
        - Return: none
        - Process: Delete the account passed in
    - void appendAccount(accountList *list, account *acct)
        - Parameter: Account linked list and account
        - Return: none

        - Process: Add an account to the end of the account list
    - account *getAccount(accountList *list, int currentPointer)
        - Parameter: Account linked list and integer current pointer
        - Return: an account
        - Process: Acquire an account with a number pointing to the position
    - int getAccountListLength(accountList *list)
        - Parameter: Account linked list
        - Return: The integer length of the account linked list passed in
        - Process: Returns the integer length of the account linked list passed in
    - void setUpAccount(account *acct, int ID, char user[20], char pass[20])
        - Parameter: Account, integer ID, username, and password
        - Return: None
        - Process: The function sets the ID, username, and password of the account passed in
    - void setSocket(account *acct, int socketNum)
        - Parameter: Account, and socket number
        - Return: None
        - Process: The function sets the socket number of the account that is passed in.
    - void setOppSocket(account *acct, int socketNum)
        - Parameter: Account, socket number of the opponent
        - Return: None
        - Process:The function sets the socket number of the account that is playing against the account that is passed in
    - int getAccountID(account *acct)
        - Parameter: Account
        - Return: Account ID integer number
        - Process:The function gets the ID of the account passed in
    - char* getAccountUser(account *acct)
        - Parameter: Account
        - Return: Account ID integer number
        - Process:The function gets the ID of the account passed in
    - char* getAccountPass(account *acct)
        - Parameter: Account
        - Return: Account password
        - Process:The function gets the password of the account passed in
    - int getAccountSocket(account *acct)
        - Parameter: Account
        - Return: Account socket number
        - Process:The function gets the socket number of the account passed in
    - int getAccountOppSocket(account *acct)

        - Parameter: Account
        - Return: Account opponent socket number
        - Process:The function gets the socket number of the opponent of the account passed in
- board.h, board.c
    - Int clientInput (int sock, char move[4], char gameBoard[8][8][2], char player)
        - Parameter: socket file descriptor, move array, gameBoard array and the player chess color.
        - Return: 0 or 1. 0 for valid input and 1 for invalid input.
        - Process: If the player has exited the game or not
    - void writeBoard(int sock, char gameBoard[8][8][2])
        - Parameter: socket file descriptor and gameBoard array. It holds the current state of the board in where all the chess pieces are.
        - Return: none
        - Process:Print the board of the game on the client terminal from the server

For remaining functions and parameters, refer to the chess version 1.0 software specifications.

3.3. Detailed description of input and output formats

3.3.1. All data sent to the server will be either a valid move or a valid message, all error handling will be client side.

3.3.2. Messaging Format (in game while playing)

3.3.2.1. -m <message text>

3.3.3. Move Format

3.3.3.1. -a <username> <move>

3.3.4. Login Format

3.3.4.1. -l <username> <password>

4. Development plan and timeline

4.1. Partitioning of tasks

- Client and server
- Login and Registration
- Send and Receive data
    - Messages and game input moves

4.2. Team member responsibilities

5/23-5/30	**Jacob-** Login System **Lichen -** menu update **Ryan -** Login system **Nghi -** Send and Receive inputs **Rayi -** Write test client and server
5/31-6/6	**Jacob-** Server game runner **Lichen -** Server game runner **Ryan -** Server game runner **Nghi -** Server game runner **Rayi -** rewrite the client to new spec

Back Matter

Copyright

References

Contributors: Jacob Bokor
Rayi Lam
Ryan Shum
Nghi Nguyen
Lichen Wang

Affiliation: Henry Samueli School of Engineering
University of California, Irvine

Index