King's Gambit Chess V2 Developer Manual

KGChess V2

Team 20:
King's Gambit

Erick Mercado
Tan Huynh
Samuel Briones-Plascencia
Thanh Tran
Larrenz Carino

UC Irvine EECS 22L Spring 2021

Table of Contents

Glossary

Term	Definition
Access Control List (ACL)	Applies rules to switch ports or (Internet Protocol) IP addresses available to a host or switch.
Algorithm	A procedure or formula used for solving a problem. It is based on conducting a sequence of specified actions in which these actions describe how to do something
ASCII (American Standard Code for Information Interchange)	A character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices.
Array	A data structure consisting of a collection of elements, each identified by at least one array index or key.
Bandwidth	the channel occupied by the carrier and measured as the difference in hertz (Hz) between the highest and lowest frequencies in the channel.
Char	It stores a single character and requires a single byte of memory in almost all compilers.
Client/server architecture	When servers provide one or more possible resources to less powerful systems, known as clients, upon request.
Cryptography	The practice and study of techniques for secure communication in the presence of third parties called adversaries.
Data	Any form of information that is used as input to a system or application for processing or that is produced as the output from a system or application.
Data Types	A particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.
Executable	File that contains a program and is capable of being executed and run on a computer
Encryption	The act of altering data to make it unreadable by any person or device, except by the intended recipient

GUI (Graphical User Interface)	A computing user interface (UI) characterized by a full-screen work area and icons that represent objects such as folders and files.
Host	A networked device with an IP address; a mainframe computer system; the operating system on which a hypervisor runs.
IPv4	The primary Layer-3 protocol in use on the internet and private intranets. Provides end-to-end logical identifiers—32-bit IPv4 addresses.
IP Address	A numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication.
Linked List	File that contains a program and is capable of being executed and run on a computer
LAN (Local Area Network)	A network consisting of all devices behind and including a single router interface
Server	a computer or computer program which manages access to a centralized resource or service in a network.
TCP	Transport protocol that is used on top of IP to ensure reliable transmission of packets
tar.gz	tar is a computer software utility for collecting many files into one archive file, often referred to as a tarball, for distribution

1. Installation

1.1 System Requirements

- Recommended OS: Linux (CentOS release 6.10, Kernel: 2.6.32-754)
    - Contains a GCC/GNU compiler
- Memory: At least 512MB RAM
- Game Storage: 1.03 Megabytes

1.2 Setup and Configuration

- **Clone files from GitHub**
  git clone https://UCINETID@github.uci.edu/EECS-22L-S-21-Team-Projects/Team20.git
- **Get tar.gz archive**
  cp ~Team20/bin/KGChess.tar.gz

1.3 Building, compilation, installation

- **Extract tar.gz package**
  gtar xvzf bin/KGChess.tar.gz
- **Change current directory to KGChess**
  cd KGChess
- **Compile and generate executable**
  make all
- **Run program offline**
  bin/KGChess
- **Run program online**
  bin/KGChess HOSTNAME PORT
- **Run server**
  bin/Server PORT
- **Test for memory leaks using Valgrind**
  make memorytest
- **Remove executable and object files**
  make clean
- **Remove all files (includes .c and .h files)**
  make cleanall

2. Client Software Architecture Overview

2.1 Main Data Types and Structures

Data Types:

- **char array[256]** - The character array would be tossed back and forth between both clients through the server. The data would then be handled by the client's program except for when the user input is invalid
- **int variables:**
    - **Ex: (int domain, int type, int protocol)** - Need integer variables for opening sockets, input to functions needed in server. Both the client and server need to set up their sockets in order to start operating and connecting to each other.

2.2 Major Software Components

- **Diagram of Module Hierarchy**

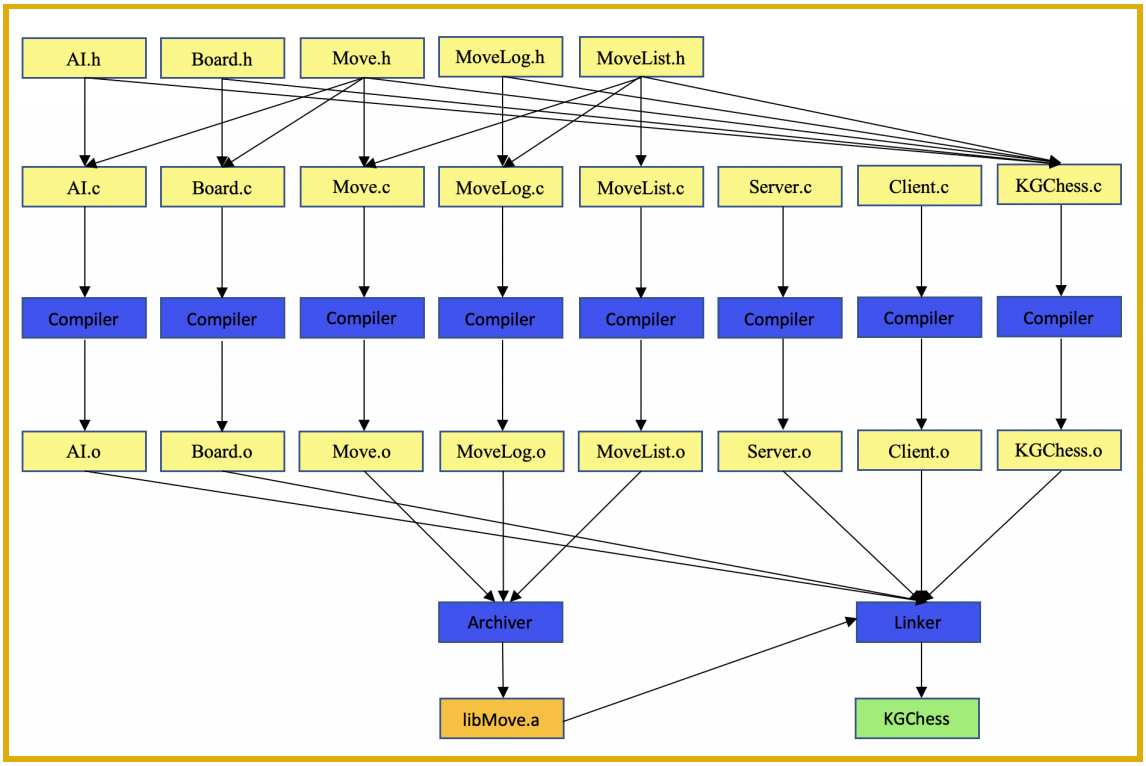

Figure 1: Diagram of how the modules will be linked in Makefile

2.3 Module Interfaces

- **API of major module functions (** only new and modified modules, for other modules, please refer to V1 software application manual **)**
  - **Board.c**
    - Updates after each input
  - **MoveLog.c**
    - Writes to log
    - Save log
    - View log
- **MoveList.c**
  - Move list
    - Doubly linked list struct implementation to store a list of moves. This list will support take back a move and move log feature
- **Client.c**
  - Client will communicate with server by sending user inputs
  - Txt file used to store account information such as username and password
  - Registering
    - User creates a username and password which gets stored on a text file

2.4 Overall Program Control Flow

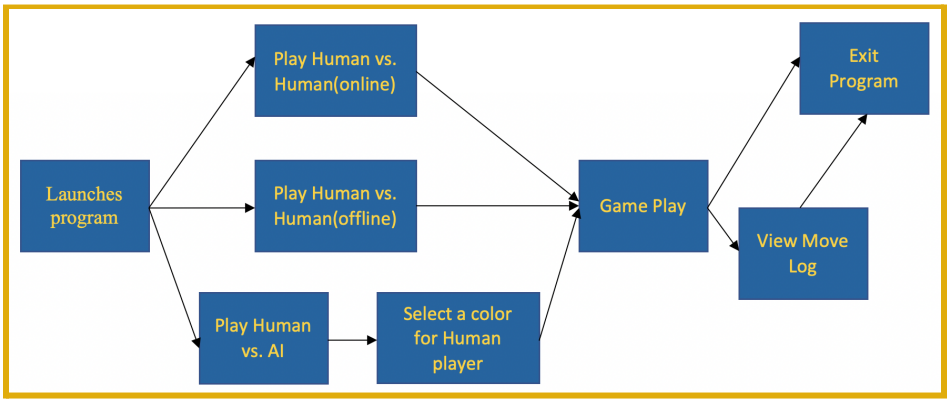

Figure 2: User flow overview

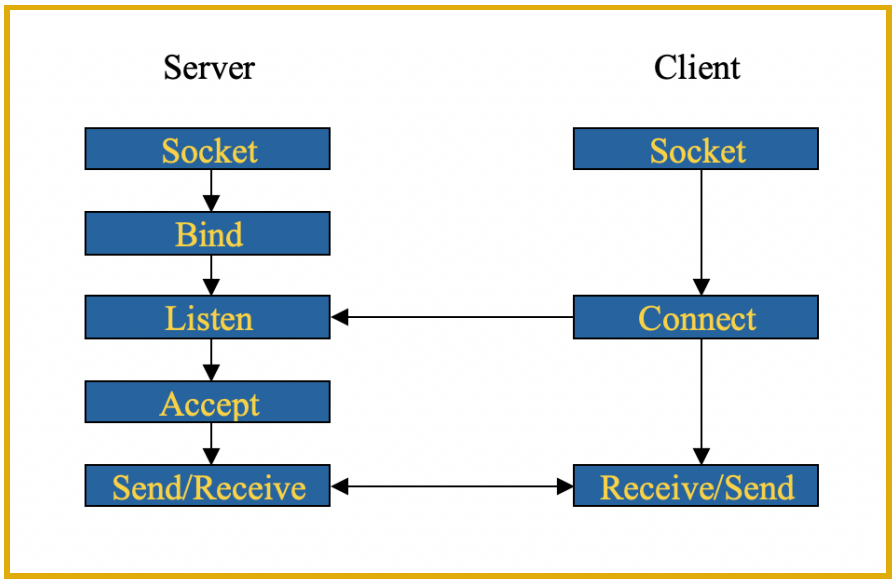

Figure 3: Server-Client interaction

3. Server Software Architecture Overview

3.1 Main Data Types and Structures

Data Types:

- **char Array [256]:** Receives the user inputs from client and sends it to the other client
- **int variables:**
  - **Ex: (int domain, int type, int protocol)** - Need integer variables for opening sockets, input to functions needed in server. Both the client and server need to set up their sockets in order to start operating and connecting to each other.

Structures:

- **These structures are needed for port server binding to a specific host and internet addressing.**
  - **Struct sockaddr{**
    unsigned short
    char

} - generally any type of family and data will have a specific address format. (protocol independent)

- **Struct sockaddr_in {**
  short **sin_family;**
  unsigned short **sin_port;**
  struct **in_addr sin_addr;**
  char **sin_zero[8];**

  } - dealing specifically with internet based communication.
- **struct in_addr {**
  unsigned long **s_addr;**

  }

3.2 Major Software Components

- **Diagram of Module Hierarchy**

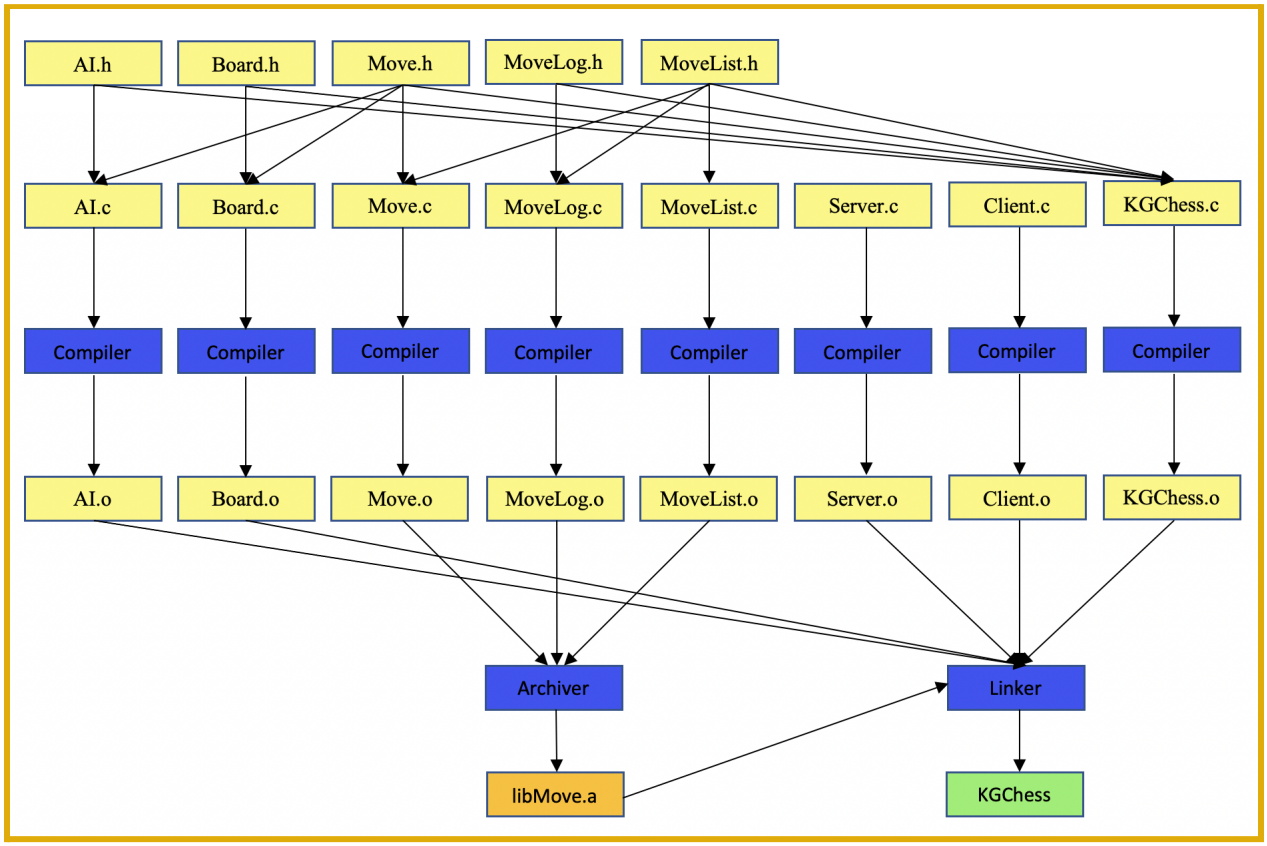

Figure 4: Diagram of how the modules will be linked in Makefile

3.3 Module Interfaces

- API of major module functions
  - Server.c
    - Servers hosted txt file to store data related
    - Server stores and send updated board data
    - Server verifies login for users and rules

3.4 Overall Program Control Flow

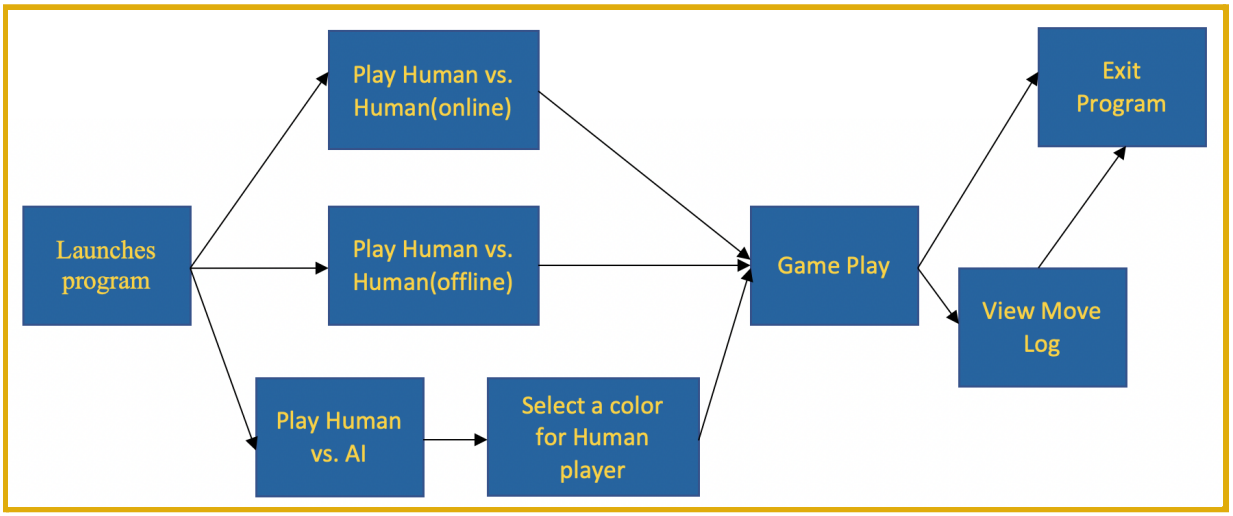

Figure 5: User flow overview

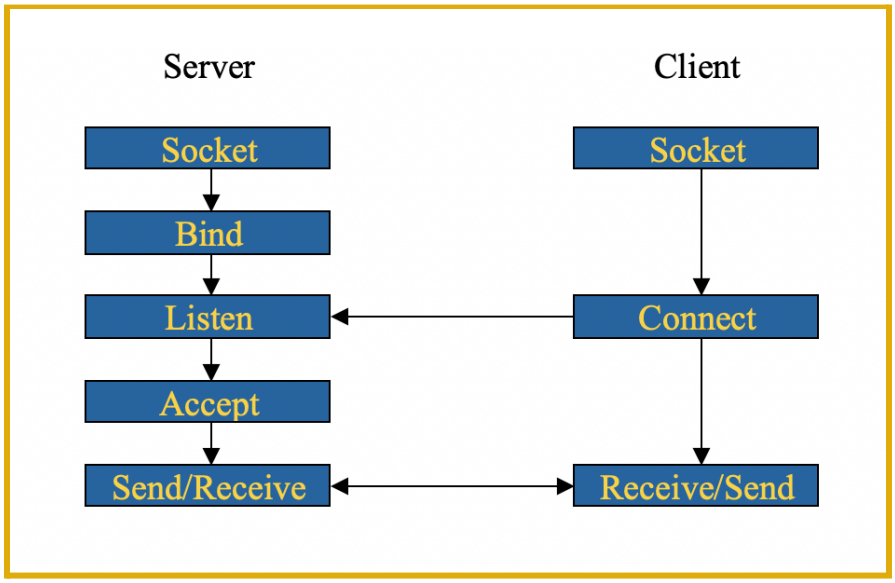

Figure 6: Server-Client interaction

4. Documentation of packages, modules, interfaces

4.1 Detailed Description of Data Structures

- **Client.c**

```
    int l, n;
    int SocketFD,   /* socket file descriptor */
    PortNo;     /* port number */
    struct sockaddr_in
    ServerAddress;  /* server address we connect with */
    struct hostent
    *Server;    /* server host */
    char SendBuf[256];  /* message buffer for sending a message */
    char RecvBuf[256];  /* message buffer for receiving a response
*/
```

Figure 7 : Example of structs, ints and functions used in client.c

- For client.c we will be using structs such as sockaddr_in and hostent.
- In addition we will be initializing SocketFD which will serve as the file description for the socket
- Two chars of size 256 will be used, SendBuff will be to buffer in order to send a message and RecvBuf will do the same for a receiving message

- **Server.c**

```
int l, n;
int ServSocketFD,/* socket file descriptor for service */
      DataSocketFD,     /* socket file descriptor for data */
      PortNo;           /* port number */
    socklen_t ClientLen;
    struct sockaddr_in
      ServerAddress,    /* server address (this host) */
      ClientAddress;    /* client address we connect with */
    char RecvBuf[256]; /* message buffer for receiving a
message */
    char SendBuf[256]; /* message buffer for sending a
response */
    int Bye = 0,
      Shutdown = 0;
```

Figure 8: Example of code in server.c

- Two chars of size 256 will be used, SendBuff will be to buffer in order to send a message and RecvBuf will do the same for a receiving message

4.2 Detailed Description of Functions and Parameters

- **Server.c**

```
void error(const char *Program, const char
*ErrorMsg)
```

Figure 9:Example code

Void error

- This function is called when a system call fails. It displays an error message and aborts the program.

```
void server(int argc, char *argv[])
```

Figure 10:Example code

Void Server

- This function has the file descriptors that store the values returned by the socket system call and the accept system call.
- It also handles the ports in which we need to use in order to handle client - server communication.
- It also calls on the socket() system to create a new socket. It takes in the domains, the type of socket, and the protocol.

```
int main(int argc, char *argv[])
```

Figure 11 :Example code

Int main

- The main function of Server.c. Calls the server function and has the parameters of argc and *argv[].

- **Client.c**

```
void My_User_Input_Game(int count, char uInpt[5])
```

Figure 12:Example code

Void My_User_Input_Game

- This function takes in the user's input for the their turn during the game.

```
void printWin(int win)
```

Figure 13:Example code
Void printWin

- This function prints the different win conditions.
  - Ex: checkmate, check, stalemate, draw.

```
int Exit_Undo(char uInpt[5], int count, int
Board[8][8], MLIST *Move_List)
```

Figure 14:Example code
Int Exit_Undo

- This function performs both the exit and the undo functions of our chess game. If the player wants to quit in the middle of a game, they input 'exit'. If the player wants to undo a move they did, they input 'undo'.

```
void error(const char *Program, const char
*ErrorMsg)
```

Figure 15:Example code
Void Error

- This function is called when a system call fails. It displays an error message and aborts the program.

```
void client(int argc, char *argv[])
```

Figure 16:Example code
Void client

- This function has the file descriptors that store the values returned by the socket system call and the accept system call.
- It also handles the ports in which we need to use in order to handle client - server communication.

        - It also calls on the socket() system to create a new socket. It takes in the domains, the type of socket, and the protocol.
        - It also handles the client address which connects to the server.

4.3 Detailed Description of the Communication Protocol

- **Communication Protocol**
    - User Selects Online Menu Option
    - This has the client try to access the server
    - Server will the accept client
    - Client will then need a login to access server for online mode
    - Log in menu appears
    - Client sends username and password
    - Server confirms correct username and password
    - Client sends users inputs for Chess game
    - Server confirms inputs
    - Server then sends board data and updates board after each user input
    - Server confirms input and will either educate it or reject it
    - Client Input and Server Confirmations and board data will repeat till game is over
    - When Game is over or a user opts to cancel the match, Server will signal "Victory ___" or "Game Ended".
    - Client displays the result
    - Users can input "Play again" and the client will signal the server and steps will be repeated.
- **Protocol messages to forward to server**
    - `login [username] [password]` - Logins to an existing account with a username and password
    - `register [username] [password]` - Registers a new account in the database with a username and password

- `chat [message]` - Sends a message to the server which will pass the message over to the other client
- `move [move input]` - Will check if the move input is valid and updates the board locally before sending to the server to update the other client's board
- `quit` - Exits the client
- `shutdown` - Exits the client and shuts down the server

5. Development Plan and Timeline

5.1 Partitioning of Tasks

- Tasks will be partitioned as evenly as possible so that each member will have a similar sized workload to prevent burnout, stress and promote a healthy work environment.
- Team Members will be able to volunteer for tasks and priority is given to members who request the task.
- In addition, experience and skill will be taken into account when assigning tasks if no member has volunteered for a given task. Example: Team Members with API experience will be asked to do the API task if no one elle volunteers since they would have the most experience in the given subject.
- Tasks will not be set in stone and Members may switch tasks or request assistance from other members and if outside issues such as sickness arise that will also be taken into account and tasks will be adjusted accordingly.

5.2 Team Member Responsibilities

- **Larrenz (Presenter):** User manual, sections 2, 3 and 4
- **Erick (Manager):** User Manual, Glossary, ToC, Index, Section 2, 4 , 5, 6
- **Tan (Recorder):** Section 1, 2, and 3 + socket integration for client/server and Makefile

- **Thanh (Reflector):** User manual, Section 1, 2, 3
- **Samuel (Reflector):** Coding Client.c and Server.c, User Manual, Section 3, and 4

6. Back Matter

6.1 Copyright

6.2 References

Dang, Quoc-Viet. "EECS 22L, SPRING 2020, UNIVERSITY OF CALIFORNIA, IRVINE." UCI Canvas. canvas.eee.uci.edu/courses/36413.

Index

A

C

D

E

F

G

I

L

O

S

T

V