CNCO3002

Advanced Computer Communications

3rd Year, 2nd Semester

Assignment

Mastermind client-server game

Submitted to

Curtin University

By

Name: Sadeep Dilshan Kasthuriarachchi

SLIIT Registration Number: IT19082066

Curtin Registration Number: 20543873

-------	18/10/2021
Signature	Date

Document version: 1.1 (2015-11·

Curtin University – Department of Computing

Assignment Cover Sheet /Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Kasthuriarachchi	Student ID:	20543873
Other name(s):	Sadeep Dilshan		
Unit name:	Advanced Computer Communications	Unit ID:	CNCO3002
Lecturer / unit coordinator:	Dr. Sie Teng Soh	Tutor:	Mr. U U Samantha
Date of submission:	17/10/2021	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* **to any other students who may gain unfair advantage from it.**
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have donemyself, specifically for this assessment. I cannot re-use the work of others, or my own previouslysubmitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Date of signature: 18/10/2021

Signature:

(By submitting this form, you indicate that you agree with all the above text.)

Table of Content

1. System Requirements ..... *- 3 -*

2.Codes ..... *- 4 -*

2.1 Server Code (MServer) ..... - 4 -

2.2 Client Code (MClient) ..... - 13 -

2.3 README File ..... - 26 -

2.4 makefile File ..... - 27 -

3. Run the Programme ..... *- 28 -*

4. Server Programme ..... *- 39 -*

4.1 Shared data structures used in the server (MServer) ..... - 39 -

5. Assumptions,Limitations and Errors ..... *- 42 -*

5.1 Assumptions ..... - 42 -

5.2 Limitations ..... - 43 -

5.3 Errors ..... - 44 -

6. Sample Inputs and Outputs of the Programs ..... *- 44 -*

6.1 Inputs and Outputs ..... - 44 -

7. References ..... *- 52 -*

Table of Figures

Figure 1.1: Fedora 34 ARM64 ..... - 3 -
Figure 2.1.1: MServer.c File ..... - 12 -
Figure 2.2.1: MClient.c File ..... - 25 -
Figure 2.3.1: README File ..... - 26 -
Figure 2.4.1: makefile File ..... - 27 -
Figure 3.1: master_mind folder ..... - 28 -
Figure 3.2: Runing the makefile ..... - 29 -
Figure 3.3: Running the MServer ..... - 30 -
Figure 3.4: Running the MClient ..... - 30 -
Figure 3.5: Welcome Interface ..... - 31 -
Figure 3.6: Join the Players ..... - 32 -
Figure 3.7: Player Status ..... - 33 -
Figure: 3.8: Send request to the player ..... - 34 -
Figure 3.9: View the send request and accept or deny ..... - 34 -
Figure 3.10: Enter the guess ..... - 35 -
Figure 3.11: See opponent guess ..... - 36 -
Figure 3.12: Player timeout lost ..... - 37 -
Figure 3.13: Player timeout win ..... - 37 -
Figure 3.14: Player quit the game ..... - 38 -
Figure 3.15: Opponent win ..... - 38 -
Figure 4.1.1: Shared Data Structures in MServer ..... - 39 -
Figure 6.1.1: MServer Inputs ..... - 44 -
Figure 6.1.2: MClient Inputs ..... - 44 -
Figure 6.1.3: MServer view after Client connection ..... - 45 -
Figure 6.1.4: “help” command input and its output ..... - 45 -
Figure 6.1.5: “who” command input and output (see the available players) ..... - 46 -
Figure 6.1.6: “start” command input in player sadeep ..... - 46 -
Figure 6.1.7: received game request from player sadeep ..... - 47 -
Figure 6.1.8: Game started with nimal ..... - 47 -
Figure 6.1.9: Input the guess (player sadeep) ..... - 48 -
Figure 6.1.12: “disconnect” command in player sadeep ..... - 49 -

1. System Requirements

This Mastermind client-server game runs on the Linux based system. So, to run the below code you should have Linux based operating system.

All the Screen shots and code mentioned in this report are run on the Fedora OS. (Figure 1.1)

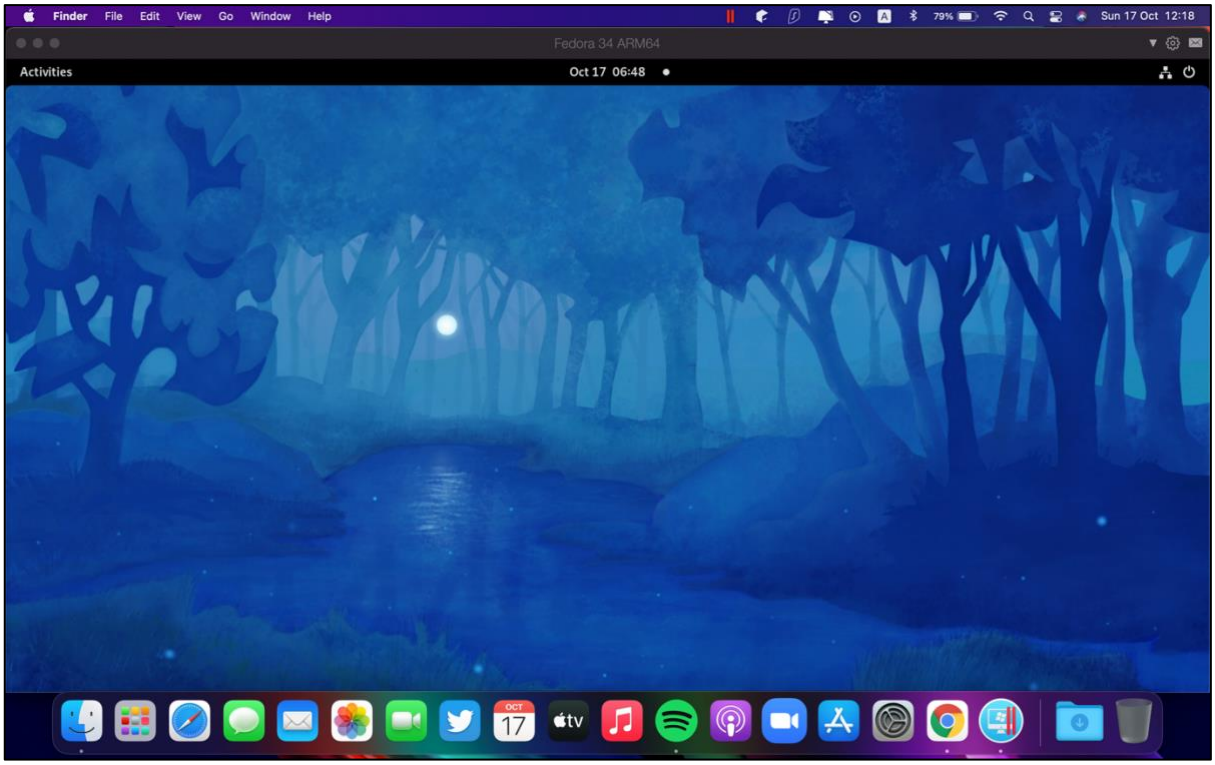

Figure 1.1: Fedora 34 ARM64

2.Codes

2.1 Server Code (MServer)

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <arpa/inet.h>
#include <sys/time.h>

// DEFINITIONS
#define MAXPENDING 10
#define BUSY 1
#define FREE 0


struct user {
   char username[32];
   char usernameAvv[32];
    short int portUDP;
   int socket;
   unsigned long AddressIP;
   int state; //BUSY or FREE
   int step;  //from 0 to 10
   int length;
   struct user *opponent;
   struct user *pointer; // next item in the user list
};

struct user* new_one=0; // new user list, consisting of a maximum of 1
element
struct user* connected_users=0;
struct user* provisional=0;
int n_connected_users=0;
struct user *client=0; // current client

struct sockaddr_in serveraddr;
struct sockaddr_in clientaddr;
char command;       //   use this command
int length;
int listener;
fd_set master;
fd_set read_fds;
int fdmax;




//FUNCTIONS FOR LIST MANAGEMENT
void remove_item_from_list(struct user* elem) { // remove from the list of
connected users
   struct user* temp=connected_users;
   if(n_connected_users==0)
```

```
      return;
   if(temp==elem) {   //This is the first on the list
        connected_users=connected_users->pointer;
      free(temp);
        n_connected_users--;
      return;
   }
   while(temp->pointer!=elem && temp->pointer!=NULL)  //scrolls the list
      temp=temp->pointer;
   if(temp->pointer==NULL)    //did not find it
      return;
    n_connected_users--;
   temp->pointer=temp->pointer->pointer;
   return;
}

void remove_from_provisional(struct user* elem) {
   struct user* temp=provisional;
   if(temp==elem) {   //Get the first on the list
        provisional=provisional->pointer;
   return;
   }
   while(temp->pointer!=elem && temp->pointer!=NULL)  //scrolls the list
      temp=temp->pointer;
   if(temp->pointer==NULL)
      return;
   temp->pointer=temp->pointer->pointer;
   return;
}


struct user* find_conn_from_socket(int sd) {
   struct user* temp=connected_users;
   while(temp!=NULL) {
      if(temp->socket==sd)
         return temp;
      temp=temp->pointer;
   }
   return NULL;
}

struct user* find_prov_from_socket(int sd) {
   struct user* temp=provisional;
   while(temp!=NULL) {
      if(temp->socket==sd)
         return temp;
      temp=temp->pointer;
   }
   return NULL;
}


struct user* find_from_username(char* name) {
   struct user* temp=connected_users;
   while(temp!=NULL) {
      if(strcmp(temp->username,name)==0)
         return temp;
      temp=temp->pointer;
   }
   return NULL;
}
```

```
int exists_conn(char* name) { // RETURN 1 IF THE NAME (OR) EXISTS AMONG THE
CONNECTED USERS
   struct user* temp=connected_users;
   while(temp!=NULL) {
      if(strcmp(temp->username,name)==0)
         return 1;
      temp=temp->pointer;
   }
   return 0;
}

//TCP COMMUNICATION FUNCTIONS

int sendTCPNumber ( int socket,int j) {
   int n ;
   n = send(socket , (void*) &j ,sizeof(int) , 0 ) ;
   if(n < sizeof(int)) {
      printf("Command error send()\n");
      exit(1);
   }
   else return n ;
}

int sendTCP ( int socket,char* buffer,int length) {
   int n ;
   n = send(socket , (void*) buffer ,length, 0 ) ;
   if ( n < length ){
      printf("Command error send()\n");
      exit(1);
      }
   else return n ; // returns number of bytes (characters) sent
}

int receiveTCPNumber ( int socket , int * number ) {
   int n ;
   n = recv(socket , (void*) number, sizeof(int) , 0 ) ;
   if ( n < sizeof(int) ){
      printf("Command error recv() \n");
      exit(1);
      }
   else return n ;
}

int receiveTCP ( int socket , char * buffer , int length ) {
   int n ;
   n = recv(socket , (void*) buffer, length , 0 ) ;
   if ( n < length ){
      printf("Command error recv()\n");
      exit(1);
      }
   else return n ;
}




//FUNCTIONS THAT IMPLEMENT THE COMMANDS AVAILABLE IN THE SERVER

void who() {
   int length;
```

```
   struct user* temp=connected_users;
   //tells the client how many users are connected
    sendTCPNumber(client->socket, n_connected_users ) ;
   while(temp) {
      length=strlen(temp->username);
      sendTCPNumber(client->socket,length);  //length
      sendTCP(client->socket,temp->username,length);    //username
      sendTCPNumber(client->socket,temp->state);    //user status
      temp=temp->pointer;
   }
}

void disconnect(char cmd) { // disconnected from the game
   printf("%s disconnected from the game \n",client->username);
   printf("%s is free\n", client->username);
   if(cmd!='k') { // it disconnected (opponent)
      printf("%s disconnected from the game \n",client->opponent-
>username);
      printf("%s is free\n", client->opponent->username);
      client->opponent->state=FREE;
      client->opponent->opponent=NULL;

   }
   client->state=FREE;
   client->opponent=NULL;
}

void start1() { // Get the length of the opponent's name
   receiveTCPNumber(client->socket,&client->length);
   client->step=5;
}
void start2() { // I get opponent name and update structures of the 2
opponents (Make them point to each other and put them BUSY)
   char cmd;
   //length of username you want to connect to
   client->step=0;
   receiveTCP(client->socket,client->usernameAvv,client->length) ;
   client->usernameAvv[client->length]='\0';
   //the client says it is responding to the connect command
   if(!exists_conn(client->usernameAvv)) {
      printf("User %s not found\n",client->usernameAvv);
      cmd='i'; // non-existent user command
      sendTCP(client->socket,&cmd,sizeof(char));
      return;
   }
   ////SEND TO THE RECIPIENT OF THE REQUEST INFORMATION ON THE CLIENT WHO
WANTS TO PLAY WITH HIM
   client->opponent=find_from_username(client->usernameAvv);
   if(client->opponent->state== BUSY) {
      printf("User %s busy\n",client->usernameAvv);
      cmd='b'; // busy user command
      sendTCP(client->socket,&cmd,sizeof(char));
      return;
   }
   //command chosen to forward the request to the client
   client->state= BUSY;
   client->opponent->state= BUSY;
   client->opponent->opponent=client;
   length=strlen(client->username);
   sendTCPNumber(client->opponent->socket,length);    //username length
   sendTCP(client->opponent->socket,client->username,length); //username
```

```
}

void start3(char cmd) { // challenge request recipient response management

    short int port;
   switch(cmd) { // switch that manages the choice of the queried client
      case 'a': { //accept
         sendTCP(client->opponent->socket,&cmd,sizeof(char));
         //sending client port
         port=htons(client->portUDP);
         int ret=send(client->opponent->socket,(void
*)&port,sizeof(port),0);// Do a normal send because it is a short int and
have not defined a function for this type
         if ( ret < sizeof(port)){
            printf("Command error send()");
            exit(1);
            }
          ret=send(client->opponent->socket,(void *)&client-
>AddressIP,sizeof(client->AddressIP),0); //  Do a normal send because it is
an unsigned long.
         if ( ret < sizeof(client->AddressIP)){
            printf("Command error send()");
            exit(1);
            }
         printf("%s connected to %s\n",client->username,client->opponent-
>username);
         break;
      }
      case 'r':  { //refused
         sendTCP(client->opponent->socket,&cmd,sizeof(char));
         client->state=FREE;
         client->opponent->state=FREE;
         client->opponent->opponent=NULL;
         client->opponent=NULL;
         break;
      }
      //there is no default case
   }
}

void quit() {

   printf("The client %s has disconnected from the server\n",client-
>username);
   close(client->socket);
   FD_CLR(client->socket,&master); // Set to 0 the bit concerning the
client, so it is not checked if it is ready.
    remove_item_from_list(client); // Remove from the list of connected
users
}

void initialize1(){
   receiveTCPNumber(client->socket,&length); // Get name length
   client->length=length;
   client->step=2;
}

void initialize2() {   //Get the name

   receiveTCP(client->socket,client->username,length);
   client->username[client->length]='\0';//string management: end-of-string
```

```
character
   client->step=3;
}

void initialize3(){    //here a user is permanently removed from the
provisional list and placed with the connected ones
   char cmd='@';
   int ret ;
   ret = recv(client->socket,(void *)&client->portUDP,sizeof(client-
>portUDP),0); // PORT UDP is short int
   if ( ret < sizeof(client->portUDP)) {
      printf("Command error recv() \n");
      exit(1);
      }
   client->portUDP=ntohs(client->portUDP);
    remove_from_provisional(client); // Remove from the provisional list
   if(exists_conn(client->username)) { // it is 1 if a user with that
username already exists
      cmd = 'e'; //exists_conns
      sendTCP(client->socket,&cmd,sizeof(cmd));
      printf("The client used an invalid username!\n");
      close(client->socket);
      FD_CLR(client->socket,&master); // Set to 0 the bit concerning the
client, so it is not checked if it is ready.
      free(client);// free memory
      client=NULL;
      return;
   }
   client->pointer=connected_users; // Put it at the top of the list
    connected_users=client; // Update head pointer of related list
    n_connected_users++;
   sendTCP(client->socket,&cmd,sizeof(cmd));
   client->step=0;
   printf("%s connected\n",client->username);
   printf("%s is free\n",client->username);
}


//////////////////START OF MAIN
int main(int argc, char* argv[]) {
   int i;
   int addrlen; //Address length
   int des_conn_sock;  //connected socket descriptor
   int yes=1;
   int ret ; // variable for returns
   if(argc!=3) { //Number of parameters
      printf("Error when passing parameters\n");
      exit(-1);
      }
    ret=atoi(argv[2]);
   if(ret<1024||ret>65535) {
      printf("Error in the chosen port number\n");
      exit(1);
   }
   memset(&serveraddr,0,sizeof(serveraddr));
   serveraddr.sin_family=AF_INET; // IPv4 internet protocols
   serveraddr.sin_port=htons(ret); // convert to network order format
   ret=inet_pton(AF_INET,argv[1],&serveraddr.sin_addr.s_addr); // From
presentation to numeric format
   if(ret==0) {
      printf("Invalid Address!!\n");
```

```
      exit(1);
   }
   printf("Street address: %s (Port: %s)\n",argv[1],argv[2]);
   listener=socket(AF_INET,SOCK_STREAM,0); // socket TCP
   if(listener==-1){
      printf("socket creation error (server)\n");
      exit(1);
   }
   if(setsockopt(listener,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==-1) {
      printf("Error during the setsockopt()\n");
      exit(1);
   }
   ret=bind(listener,(struct sockaddr *)&serveraddr,sizeof(serveraddr)); //
sockaddr_in ->sockaddr
   if(ret==-1){
      printf("error during the bind(): maybe a server is already up\n");
      exit(1);
   }
   ret=listen(listener,MAXPENDING);
   if(ret==-1){
      printf("error in listen()\n");
   exit(1);
   }
   FD_ZERO(&master); // Initialize the set
   FD_ZERO(&read_fds); // Initialize the set
   FD_SET(listener,&master); // Set the bit related to listener, server, to
1
   fdmax=listener; // The one with the highest index, for now listener is
the only one
   for(;;){      //infinite loop
      read_fds=master; // Copy of the descriptors to check because the
input set is modified
      ret=select(fdmax+1,&read_fds,NULL,NULL,NULL);
      if(ret==-1) {
         printf("error during the select()\n");
         exit(1);
      }
      for(i=0;i<=fdmax;i++) {
         if(FD_ISSET(i,&read_fds)) { // Check if a descriptor is ready
            if(i==listener) { // If it is the listening socket
               addrlen=sizeof(clientaddr);
               des_conn_sock=accept(listener,(struct sockaddr
*)&clientaddr,(socklen_t *)&addrlen);
               if(des_conn_sock==-1)  {
                  printf("Error during the accept()\n");
                  exit(1);
               }
               printf("Connection established with the client\n");
               FD_SET(des_conn_sock,&master); // Set the descriptor in the
master
               if(des_conn_sock>fdmax) fdmax=des_conn_sock; // Choose the
upper extreme
                     new_one=malloc(sizeof(struct user)); // Put the user
just logged in at the top of the "new" list
                     new_one->AddressIP=clientaddr.sin_addr.s_addr;
                     new_one->socket=des_conn_sock;
                     new_one->state=FREE;
                     new_one->opponent=NULL;
                     new_one->step=1;
                     new_one->pointer=provisional;
                     provisional=new_one; // Put at the top of the
```

```
"provisional" list
                    new_one=NULL;
            }  //case closure i==listener
            else {
               client=find_prov_from_socket(i);
               if(client!=NULL) {        // found one

                  if(client->step==1)
                           initialize1();
                  else if(client->step==2)
                           initialize2();
                  else if(client->step==3)
                           initialize3(); // This is where take him the
provisional list because he has registered completely
               }
               else {
                  client=find_conn_from_socket(i); // Pass the socket and
go to look in the list of users_connected
                  if(client == NULL) {
                     printf("The client does not exist\n");
                     exit(1);

                  }
                  if(client->step!=0) {
                     switch(client->step) {
                        case 4: {
                           start1(); // see comment start1()
                           break;
                        }
                        case 5: {
                           start2(); // see comment start2()
                           break;
                        }
                     }
                  }
                  else   { // case step == 0
                     ret=recv(i,(void *)&command,sizeof(command),0);
                     if(ret==0) {
                        quit(); // Disconnection due to inactivity
                     }
                         else  if ( ret < sizeof(command)) {
                        printf("Command error recv()\n");
                        exit(1);
                        }
                     switch(command) {
                        case 'r':
                        case 'a': {
                           start3(command); //see comment start3()
                           break;
                        }
                        case 'w': {
                           who();
                           break;
                        }
                        case 'k': case 't':
                        case 'd': {
                           disconnect(command);
                           break;
                        }
                        case 's': { // !start
                           client->step=4;
```

```
                            break;
                        }
                        case 'q': {
                            quit();
                            break;
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

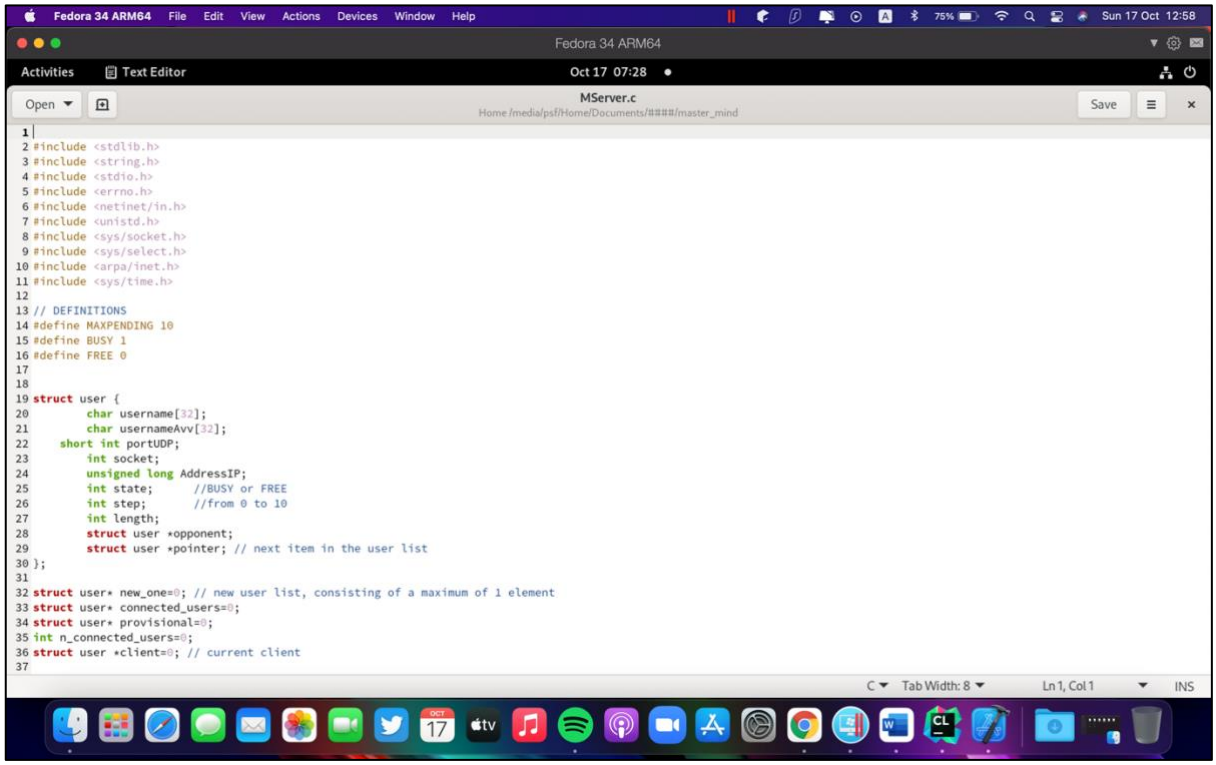

Figure 2.1.1: MServer.c File

2.2 Client Code (MClient)

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <arpa/inet.h>
#include <sys/time.h>

//DEFINITIONS
#define COMBLEN 4 //combination length
#define BUSY 1
#define FREE 0
#define TURNOAVV 0
#define MYTURN 1

int socketS;
int socketC;

struct sockaddr_in serveraddr;
struct sockaddr_in avvaddr; // adversary address
struct sockaddr_in myaddr;
struct timeval timer;
int addrlen ;
int num_correct = 0 ; // num. correct and in the right place
int num_wrong = 0 ; // num. correct but in the wrong place

//REFERENCE CLIENT
char myUsername[32];
unsigned long myAddressIP;
short int PortUDP;
int myComb ;
int myAttempt ;

//OPPONENT
char usernameAvv[32];
unsigned long addressIPAvv;
short int portUDPAvv;
// Memorize the combination proposed by the opponent
int attemptAvv  ;

//opponent
int shift=1;
int n_connected_users;
char commands[6][30] = {
   "!help",
   "!who",
   "!quit",
   "!start",
   "!disconnect",
   "!combination"
};

//VARIOUS
char shell;
```

```
fd_set master;
fd_set read_fds;

int fdmax;
int length;
char help[]=("\n\n\n                                                    ✪ ✪
✪ ✪ ✪ ✪ ✪ ✪ ✪ ✪ WELCOME TO THE MASTERMIND CLIENT-SERVER GAME ✪ ✪ ✪ ✪ ✪
✪ ✪ ✪ ✪ ✪\n\n\n"
             " ♛ The mastermind will choose a secret code from a set of
given symbols.\n"
             " ♛ Your job is to try and guess the code. For each guess you
make the mastermind.\n"
             " ♛ We tell you how many of your guesses are the right
symbol.\n"
             " ♛ We tell you how many are the right symbol, but in the
wrong location.\n"
             " ♛ You have 10 guesses to try and outwit the
mastermind.\n\n"
             " ♛ The valid symbols are: \n\n"
             "      🎱  Cyan    =  C  =  0\n"
             "      🎱  Green   =  G  =  1\n"
             "      🎱  Magenta =  M  =  2\n"
             "      🎱  Orange  =  O  =  3\n"
             "      🎱  Blue    =  B  =  4\n"
             "      🎱  Purple  =  P  =  5\n"
             "      🎱  Red     =  R  =  6\n"
             "      🎱  Silver  =  S  =  7\n"
             "      🎱  White   =  W  =  8\n"
             "      🎱  Yellow  =  Y  =  9\n\n"
             " ⚠  ALERT : One time player can only use 4 valid symbols
(i.e CGMO or 0123). It should be 4 digit value.  ⚠ \n"
             " \n
☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻__Good luck__☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻☻\n"
             " \n ☛ The following commands are available:\n\n"
             " * !help        --> show list of available commands\n"
             " * !who         --> show list of clients connected to
server\n"
             " * !start       --> start a game against client_name\n"
             " * !disconnect  --> disconnects client from other peer
(opponent)\n"
             " * !quit        --> disconnects client from server\n"
             " * !combination --> lets the player try to guess (comb) the
opponent's combination\n ");

int step=0; // look challenging or I write from the keyboard


//TCP and UDP FUNCTIONS
int sendTCPNumber ( int socket,int j) {
   int n ;
   n = send(socket , (void*) &j ,sizeof(int) , 0 ) ;
   if(n < sizeof(int)) {
      printf("Command error send()\n");
      exit(1);
   }
   else return n ;
}

int sendTCP ( int socket,char* buffer,int length) {
```

```
   int n ;
   n = send(socket , (void*) buffer ,length, 0 ) ;
   if ( n < length ){
      printf("Command error send()\n");
      exit(1);
      }
   else return n ; // returns number of bytes (characters) sent
}

int receiveTCPNumber ( int socket , int * number ) {
   int n ;
   n = recv(socket , (void*) number, sizeof(int) , 0 ) ;
   if ( n < sizeof(int) ){
      printf("Command error recv() \n");
      exit(1);
      }
   else return n ;
}

int receiveTCP ( int socket , char * buffer , int length ) {
   int n ;
   n = recv(socket , (void*) buffer, length , 0 ) ;
   if ( n < length ){
      printf("Command error recv()\n");
      exit(1);
      }
   else return n ;
}

int sendUDPNumber(int socket , int * buffer ) {
      int n ;
      n = sendto(socketC, (void*)buffer , sizeof(int) , 0 , (struct
sockaddr*) &avvaddr,sizeof(avvaddr));
      if ( n < sizeof(int) ){
      printf("Command error sendUDPNumber() \n");
      exit(1);
      }
   else return n ;
}

int sendUDP(int socket , char * buffer , int length) {
      int n ;
      n = sendto(socketC, (void*)buffer , length , 0 , (struct sockaddr*)
&avvaddr,sizeof(avvaddr));
      if ( n < length){
      printf("Command error sendUDP \n");
      exit(1);
      }
   else return n ;
}

int receiveUDPNumber( int socket , int * buffer) {
      int n ;
      n = recvfrom(socketC, (void*)buffer ,sizeof(int) , 0 , (struct
sockaddr*) &avvaddr,(socklen_t*)&addrlen);
      if ( n < sizeof(int) ){
      printf("Command error recvfrom() \n");
      exit(1);
      }
   else return n ;
}
```

```
int receiveUDP( int socket , char * buffer , int length ) {
      int n ;
      n = recvfrom(socketC, (void*)buffer ,length , 0 , (struct sockaddr*)
&avvaddr,(socklen_t*)&addrlen);
      if ( n < length ){
      printf("Command error recvfrom() \n");
      exit(1);
      }
   else return n ;
}

//DEFINITION OF FUNCTIONS
void insert_combination(int * combination) {
   int ret ;
   fflush(stdin);
   printf("Type in your combination:");
   ret = scanf("%d",combination);
   if ( *(combination) > 9999 || *(combination) < 0000 || ret == 0 ) {
        printf("Invalid Combination \n" ) ;
        scanf("%*[^\n]") ;
        insert_combination(combination) ;
        return ;
    }
   if ( *combination == 0 ) printf("0000");
   else if(*combination < 10 ) printf("000%d",*combination);
   else if (*combination < 100) printf("00%d",*combination);
   else if(*combination < 1000) printf("0%d",*combination);
   else printf("%d",*combination);
   printf("\n");
}

void disconnect(int end){
   char cmd;
   if(shift==TURNOAVV && end==1){
      printf("it's not your turn\n");
      return;
   }
   switch (end) {
      case 0: { // over for the right combination
         cmd='k';
         break;
         }
      case 1: { // abandonment
         cmd='d'; // in the altar
         printf("You gave up. YOU LOST\n");
         sendUDP(socketC,&cmd,sizeof(char));
         break;
      }
      case 2: { // timeout
         cmd='t'; // main
         printf("TIMEOUT !! YOU LOST\n");
         sendUDP(socketC,&cmd,sizeof(char));
         break;
      }
   }
   printf("Disconnected from %s \n",usernameAvv);
   sendTCP(socketS,&cmd,sizeof(char));
   shell='>'; // shell command
   memset(&avvaddr,0,sizeof(avvaddr)); // Reset the structure avaddr
   return ;
```

```
}

// Check the comb that received from the opponent
void check_comb_avv(int attempt ) {
    num_correct = 0 ;
    num_wrong = 0 ;
   int ret;
   int temp1[4] ;
   int temp2[4] ;
   int rest1=attempt ;
   int rest2=myComb ;
   int i ;
   int j ;
   temp1[0] = rest1/1000 ;
   rest1 = rest1%1000 ;
   temp1[1] = rest1/100 ;
   rest1 = rest1%100 ;
   temp1[2] = rest1/10 ;
   rest1 = rest1%10 ;
   temp1[3] = rest1 ;
   temp2[0] = rest2/1000 ;
   rest2 = rest2%1000 ;
   temp2[1] = rest2/100 ;
   rest2 = rest2%100 ;
   temp2[2] = rest2/10 ;
   rest2 = rest2%10 ;
   temp2[3] = rest2 ;
   if ( attempt == myComb ) {
         int k = 4 ;
         printf("YOU LOST!!\n");
         sendUDPNumber( socketC , &k ) ;
         disconnect(0);
         return ;
      }
   for ( i = 0 ; i < 4 ; i++ ) {
      if ( temp1[i] == temp2[i] )num_correct++ ;
      else for ( j = 0 ; j < 4 ; j++) {
         if ( temp1[i] == temp2[j] ){
                  num_wrong++ ;
               break;
               }
         }
      }
   printf("%s,guess ",usernameAvv) ;
   if ( attempt == 0 ) printf("0000");
   else if(attempt < 10 ) printf("000%d",attempt);
   else if (attempt < 100) printf("00%d",attempt);
   else if(attempt < 1000) printf("0%d",attempt);
   else printf("%d",attempt);
   printf(" His attempt is wrong\n") ;
   ret = sendto(socketC,(void *)&num_correct,sizeof(int),0,(struct
sockaddr*)&avvaddr, (socklen_t)sizeof(avvaddr));
   if ( ret < sizeof(int)) {
      printf("error in sendto()");
      exit(1);
      }
      printf("it's your turn\n") ;
}

void send_answer2() {
   sendUDPNumber(socketC,&num_wrong);
```

```
   printf("I have sent the figures\n");
   shift = MYTURN; // it's my turn
   printf("it's my turn\n");
   step = 0 ;
   }

void combination () {
   char cmd = 'h' ;
   if(shell=='>') {
      printf("Command valid only in game!\n");
      fflush(stdin);
      return;
   }
   if(shift == TURNOAVV) {
      printf("Command valid only during one's turn!\n");
      fflush(stdin);
      return;
   }
    insert_combination ( &myAttempt ) ;
   sendUDP(socketC,&cmd,sizeof(char));

   sendUDPNumber(socketC,&myAttempt);
   step = 9 ;
    shift = TURNOAVV ;// it's not my turn anymore
   printf("it's the turn of %s \n", usernameAvv) ;
}

void get_replies1 () { // Get the number of the right digits in the right
place
   step = 10 ;
   char cmd ='z'; // Send the second part of the answers, see send_answer2
()
   int k ;
   receiveUDPNumber(socketC, &k) ;
   if ( k == COMBLEN ){
      printf("YOU WON!!\n") ;
      shift = MYTURN ; // it's my turn
      step = 0 ;
      disconnect(0);
      return ;
      }
    printf("%s player's guess : %d in the right place",usernameAvv,k) ;
   sendUDP(socketC , &cmd , sizeof(char));
}

void get_replies2() { // Got the right number of digits in the wrong place
   step = 0 ;
   int k ;
   receiveUDPNumber(socketC , &k ) ;
   printf("   %d right guesses in the wrong place\n",k) ;
   shift = TURNOAVV ; //it's not your turn
}

int cmdtoint (char* cmd) { // "translation" of the command in full
   int i;
   for(i=0;i<6;i++) {
      if(strcmp(cmd,commands[i])==0) return i;
   }
   return -1;
}
```

```
void start_game() {
   printf("Starting the Game\n");
   shift = MYTURN ; //it's up to you to launch the request
   step=1;
   return;
   }

void get_port(){
   int ret=recv(socketS,(void*)&portUDPAvv,sizeof(portUDPAvv),0); // Get
opponent port number
   if (ret < sizeof(portUDPAvv) ) {
      printf("Command error recv\n");
      exit(1);
   }
   step=2;
}

void receive_ip(){
   int ret=recv(socketS,(void*)&addressIPAvv,sizeof(addressIPAvv),0); //
Get opponent's ip address
   if (ret < sizeof(addressIPAvv) ) {
      printf("Command error recv\n");
      exit(1);
   }

   memset(&avvaddr,0,sizeof(avvaddr));
   avvaddr.sin_family=AF_INET; // Build the avaddr structure
   avvaddr.sin_port=portUDPAvv;
   avvaddr.sin_addr.s_addr=addressIPAvv;
   portUDPAvv=ntohs(portUDPAvv); // conversion all host byte order
   printf("Opponent port number %d\n",portUDPAvv);
   printf("%s accepted the game\n",usernameAvv);
   printf("Game started with %s\n",usernameAvv);
    insert_combination (&myComb ) ;
   printf("it's your turn:\n");
   shift = MYTURN ; // it's your turn
   step=0;
   shell = '#';
   return;
}


//FUNCTIONS THAT IMPLEMENT THE COMMANDS

void who(){
   char cmd='w';
   sendTCP(socketS,(void *)&cmd,sizeof(char));
   step=3;
   return;
}

void who0(){
   receiveTCPNumber(socketS,&n_connected_users);
    printf("%d Connected users:\n",n_connected_users);
   step=4;
   return;
}

void who1(){
   receiveTCPNumber(socketS,&length);
   step=5;
```

```
}

void who2(){
   char username[32];
   receiveTCP(socketS,username,length);
   username[length]='\0';
   printf("%s ",username);
   step=6;

}

void who3(){
   int state;
   receiveTCPNumber(socketS,&state);
   if(state == FREE) printf("FREE\n");
   else printf("BUSY\n");
    n_connected_users--;
   if(n_connected_users==0) step=0;
   else step=4; // it loops until n_connected is 0, that is, it returns to
execute who1
   return;
}

void start1() {
   int length;
   char cmd='s';
   if(strcmp(myUsername,usernameAvv)==0) {
      printf("You are choosing yourself! \n");
      return;
   }
   sendTCP(socketS,&cmd,sizeof(cmd));
   length=strlen(usernameAvv);
   sendTCPNumber(socketS,length);
   sendTCP(socketS,usernameAvv,length);
   printf("Waiting for an Answer\n"); // Waiting
   receiveTCP(socketS,&cmd,sizeof(char));
   switch(cmd) {
      case 'i': {
         printf("Unable to connect to %s: non-existent user.
\n",usernameAvv);
         break;
      }
      case 'a': {
            start_game();
         break;
      }
      case 'b': {
         printf("Can not connect: the user is busy.\n");
         break;
      }
      case 'r': {
         printf("Unable to connect to %s: the user has refused the
game.\n",usernameAvv);
         break;
      }
   }
   return;
}

void quit(){
   char cmd;
```

```
   cmd='q';
   if(shell=='#') { // In the match
      disconnect(1); // do you give up
   if(shift==TURNOAVV) return;    }
   sendTCP(socketS,&cmd,sizeof(char));
   close(socketC);
   close(socketS);
   printf("Client disconnected Successfully\n");
   exit(0);
}


//OTHER USEFUL FUNCTIONS

void read_command_in_input() {
   char cmd[100];
   fflush(stdin); //To remove any other characters from the buffer
   scanf("%s",cmd);
   switch(cmdtoint(cmd)) {
      case 0: {
         printf("%s",help);
         break;
      }
      case 1: { who(); break;}
      case 2: { quit(); break;}
      case 3: { scanf("%s",usernameAvv);
         if(shell=='>') start1();
         else printf("You're already in a Game\n");
         break;}
      case 4: { if(shell=='#') disconnect(1); else printf("You are not in a
Match\n"); break;}
      case 5: { if(shell=='#') combination(); else{ printf("You are not in
a Match\n");/*fflush(stdin);*/}; break;}
      default: printf("The command does not exist\n");
   }
   return;
}

void connect_to_server(char* addr, int port) {
   int ret ;
   memset(&serveraddr,0,sizeof(serveraddr));
    // converts the characters given by the addr buffer into an IP address
and copies them to the serveraddr
   ret=inet_pton(AF_INET, addr, &serveraddr.sin_addr.s_addr);
   if(ret==0) {
      printf("Invalid Address\n");
      exit(1);
   }

   if(port<1024||port>65535) {
      printf("Invalid port number entered!\n");
      exit(1);
   }
   serveraddr.sin_port=htons(port);
   socketS=socket(AF_INET,SOCK_STREAM,0);
   if(socketS==1) {
      printf("Error during the socket()\n");
      exit(1);
    }
    serveraddr.sin_family=AF_INET;
    ret=connect(socketS,(struct sockaddr*)&serveraddr,sizeof(serveraddr));
```

```
    if(ret==-1) {
      printf("Error during the start(). Probably the server is not
reachable.\n");
      exit(1);
   }
   return;
}

void manage0(){       // Get challenger name length
   int ret = receiveTCPNumber(socketS,&length) ;
   if(ret==0) { //The server has disconnected
      printf("The server has closed the connection.\n");
      fflush(stdout);
      exit(1);
   }
   memset(usernameAvv,0,32);
   step=7;
}


void manage1(){ // Receive a challenger username and answer
   char ris;
   char cmd;
   step=0;
   receiveTCP(socketS , usernameAvv , length) ;
   usernameAvv[length]='\0';
   printf("%s asked you to play!\n",usernameAvv);
   do {
      scanf("%*[^\n]") ;
      scanf("%c",&ris); //
      printf("Accept the request? (y|n): ");
      fflush(stdin);
      scanf("%c",&ris);
   }
   while(ris!='y'&& ris!='n');
   if(ris=='y') {
      shift=TURNOAVV; // the opponent's turn, then the first one who made
the request.
      printf("Game request accepted\n");
      printf("It is the turn of %s\n",usernameAvv);
      cmd='a';
      shell = '#';
      sendTCP(socketS , &cmd , sizeof(char));
        insert_combination(&myComb) ;
   }
   else {
      printf("Game request denied\n");
      cmd='r';
      sendTCP(socketS , &cmd , sizeof(char)) ;
   }
   return;
}



/////////START OF MAIN////////

int main(int argc,char* argv[]){
   int ret ;
   int i;
   char cmd;
```

```
    int temp;
   short int port_tmp;


   struct timeval *time;

   if(argc!=3) {
      printf("Error in passing parameters \n");
      return -1;
   }

   //The client connects to the server
    connect_to_server(argv[1],atoi(argv[2]));
   printf("Connect to the server %s (brings %s) carried out
successfully\n",argv[1],argv[2]);
   //Asks the client for data (port and username)
   printf("Insert your name: ");
   scanf("%s",myUsername);

   length=strlen(myUsername);
   sendTCPNumber(socketS , length);   //Send username length to server
   sendTCP(socketS , myUsername , length);       //Invalid username length
   do {
      printf("Enter the port number: ");
      scanf("%*[^\n]") ;
      ret=scanf("%d",&temp);
      if(temp<1024||temp>65535 || ret == 0) {printf("Wrong port number\n");
temp=1;}
      else {PortUDP=( short int)temp; temp=0;}
   }
   while(temp);
   port_tmp=htons(PortUDP);
   ret=send(socketS,(void*)&port_tmp,sizeof(port_tmp),0); //has not defined
TCPsend
   if ( ret < sizeof(port_tmp)) {
      printf("Error in send() \n");
      exit(1) ;
      }
   receiveTCP(socketS , &cmd , sizeof(char)) ;
   if(cmd=='e') {
      printf("Username already in use\n");
      exit(1);
   }
   socketC=socket(AF_INET,SOCK_DGRAM,0);
   if(socketC==-1) {
      printf("Error during the socket()\n");
      exit(1);
   }

   memset(&myaddr,0,sizeof(myaddr));
   myaddr.sin_family=AF_INET;
   myaddr.sin_port=htons(PortUDP);
   myaddr.sin_addr.s_addr=htonl(INADDR_ANY); // inaddr_any listens on any
network interface

   ret=bind(socketC,(struct sockaddr*)&myaddr,sizeof(myaddr));
   if(ret==-1){
      printf("Error during the bind(): maybe it is busy\n");
      quit();
   }
```

```
   FD_ZERO(&master); // reset the set
   FD_SET(socketS,&master); // set the server socket descriptor to 1
   FD_SET(socketC,&master); // set the socket relative to the one used to
communicate with the opponent to 1
   FD_SET(0,&master);
   fdmax=(socketS>socketC)?socketS:socketC; // Choose the upper extreme
   shell='>'; // shell command
   printf("%s",help);
   while(1) {    //loop
      read_fds=master; // Copy
      timer.tv_sec=60;
      timer.tv_usec=0;
      time=&timer;
      if(shell=='>') time=NULL; // Command mode, so the timer is not set
      if(shift==TURNOAVV) time=NULL; // Timer is not set
//    if(shell=='#') printf("I enter \n");
      if(step==0) { printf("%c",shell); fflush(stdout);}
      ret=select(fdmax+1,&read_fds,NULL,NULL,time);
//    if(shell=='#') printf("I leave the select\n");
      if(ret==-1) {
         printf("Error during the select()\n");
         exit(1);
      }
      if(ret==0) disconnect(2); // timeout, the select returns 0 if the
timeout expires
      else for(i=0;i<=fdmax;i++) {
         if(FD_ISSET(i,&read_fds)) { //ready descriptor
            if(i==0) { //Ready descriptor is stdin
               //Read command input
                     read_command_in_input();
            }
            if(i==socketS) { //ready the server
               if(step==1) get_port();
               else if(step==2) receive_ip();
               else if(step==3) who0();
               else if(step==4) who1();
               else if(step==5) who2();
               else if(step==6) who3();
               else if(step==7) manage1();
               else manage0(); // Challenge request
            }
            if(i==socketC) {
               if(step==8) {
                  step=0;
                  addrlen=sizeof(avvaddr);
                  receiveUDPNumber(socketC,&attemptAvv);
                        check_comb_avv(attemptAvv);
               }
               else if( step == 9 )
                        get_replies1() ;
               else if ( step == 10 )
                        get_replies2() ;
               else {
                  receiveUDP(socketC , &cmd , sizeof(char) ) ;
                  switch(cmd) {
                     case 'h': {
                        step=8;
                        break;
                     }
                     case 'z' : {
                                 send_answer2();
```

```
                        break;
                        }
                    case 'd': { //opponent disconnection
                        printf("%s he gave up. YOU WON!\n",usernameAvv);
                        shell='>';
                        memset(&avvaddr,0,sizeof(avvaddr));
                        break;
                    }
                    case 't': { //opponent disconnection
                        printf("timeout for %s. YOU WON!\n Disconnected
from %s\n",usernameAvv,usernameAvv);
                        shell='>';
                        memset(&avvaddr,0,sizeof(avvaddr));
                        break;
                    }
                }
            }
        }
    }
    return 0;
}
```

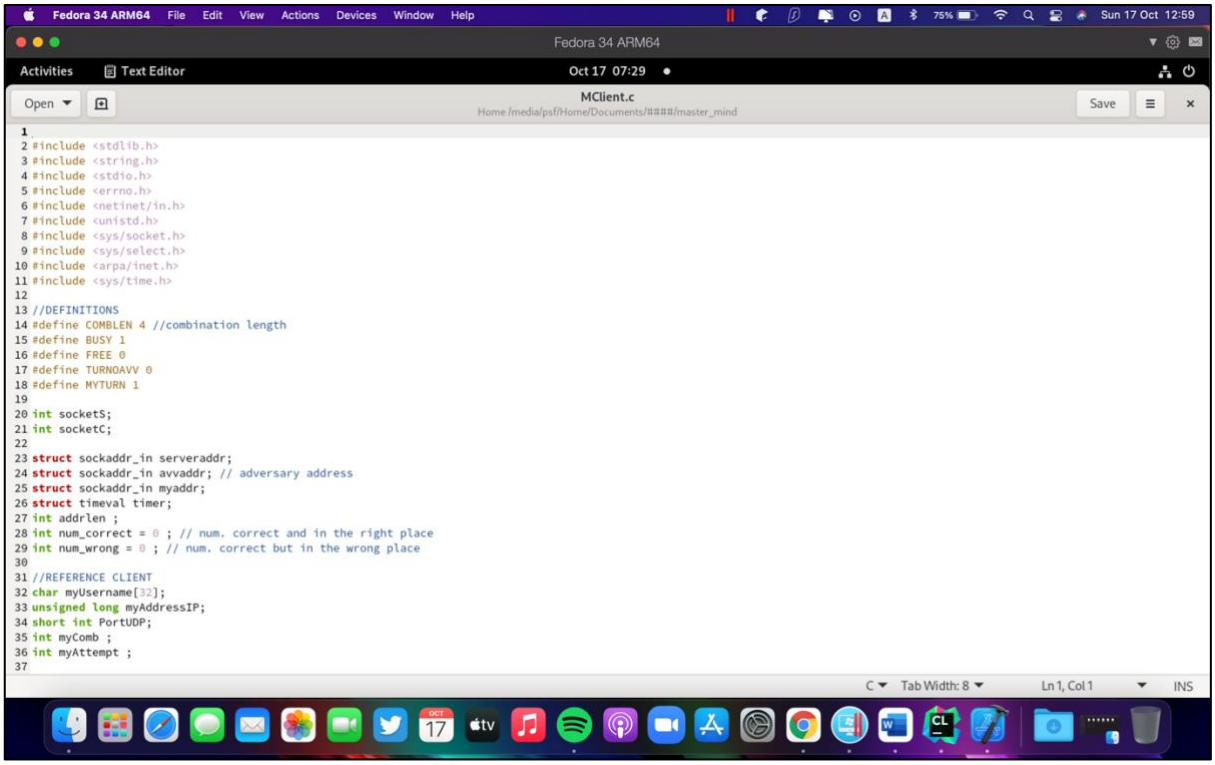

Figure 2.2.1: MClient.c File

2.3 README File

```
# Advanced Computer Communications (CNCO3002)

# Mastermind client-server game

##Compiling

Run the makefile: make -f makefile

##RUN
★ For the server, you'll need one terminal window/tab, and for each
client, you'll need one.

Run the MServer:    ./MServer 127.0.0.1 1234

★ Pass the network address and port number of the server.

Run the MClient:    ./MClient 127.0.0.1 1234

★ After that it establishes a client-server connection.

★ After that you have to enter username and the client's port number.
```

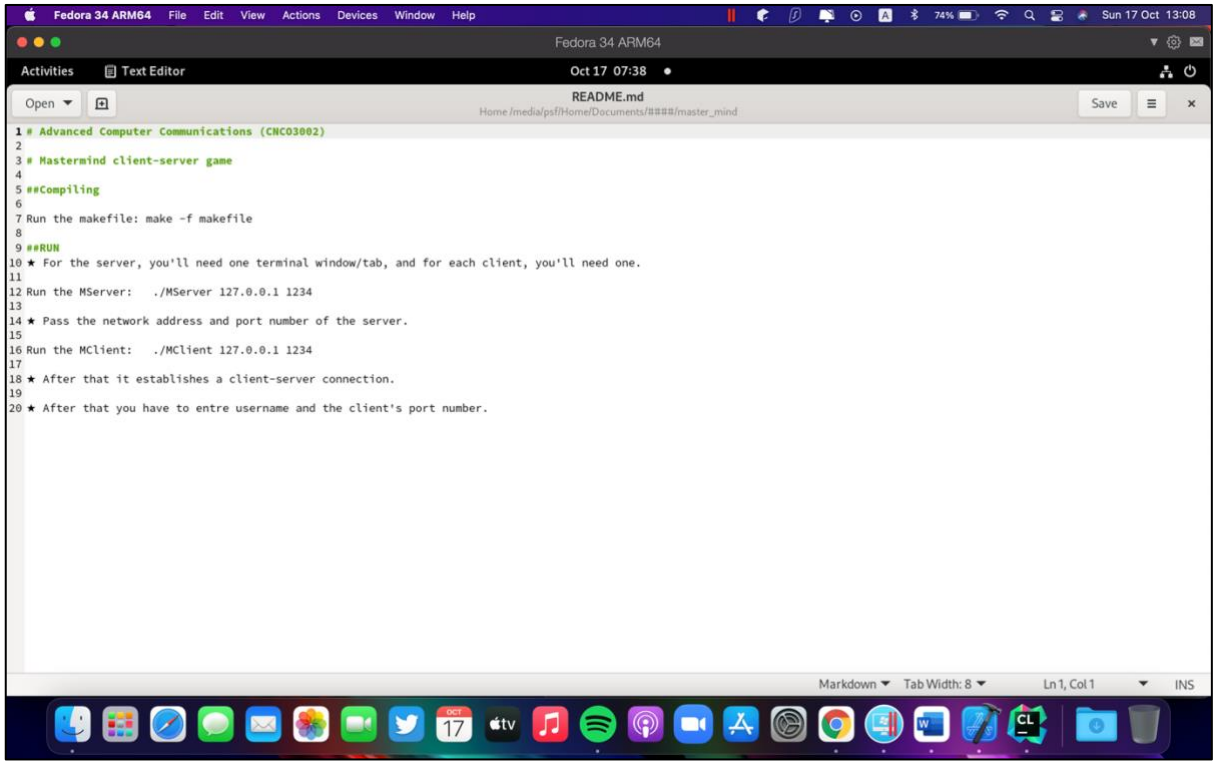

Figure 2.3.1: README File

2.4 makefile File

```
all: mastermind_server mastermind_client

mastermind_server: MServer.c
    gcc MServer.c -o MServer

mastermind_client: MClient.c
    gcc MClient.c -o MClient
```

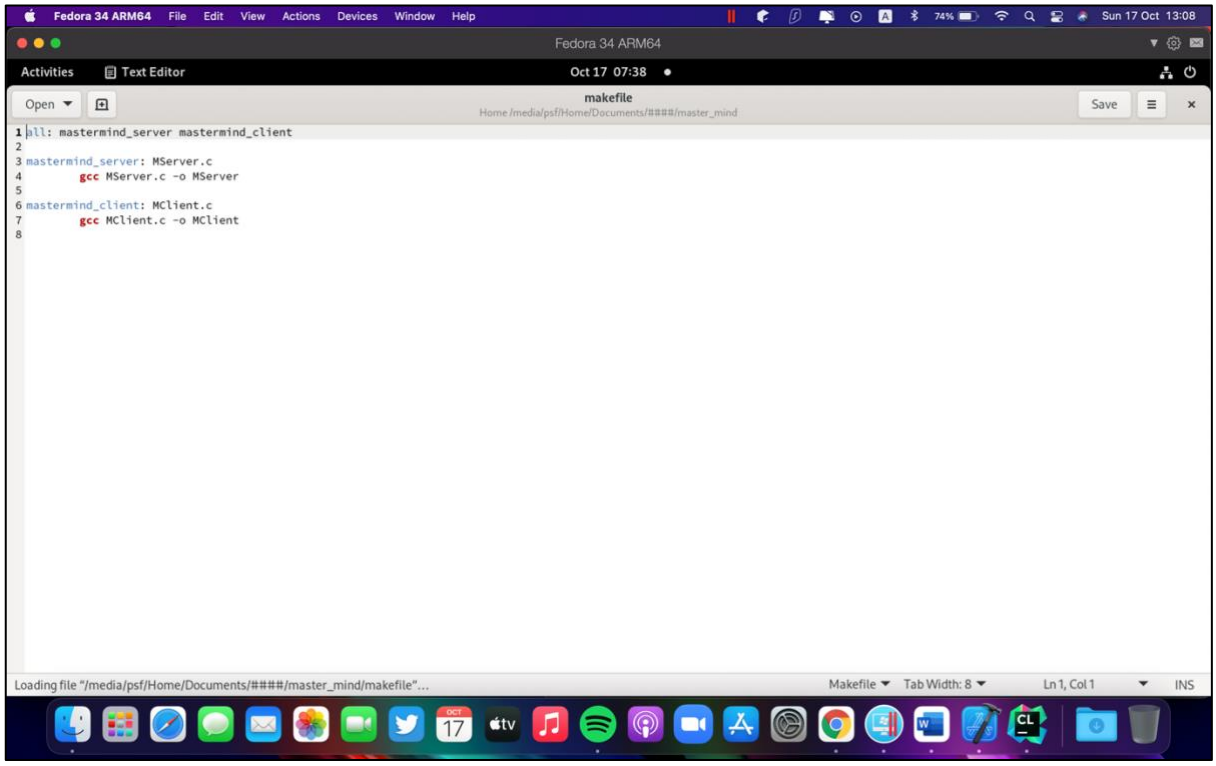

Figure 2.4.1: makefile File

3. Run the Programme

After creating the above mentioned code, files now you can run the programme using the fedora terminal. First go to the directory where the code files are located and then open the terminal.

Give the correct location where the files are located using “cd” command in the terminal. Then run the “makefile” file using the following command (figure 3.2).

makefile: make -f makefile

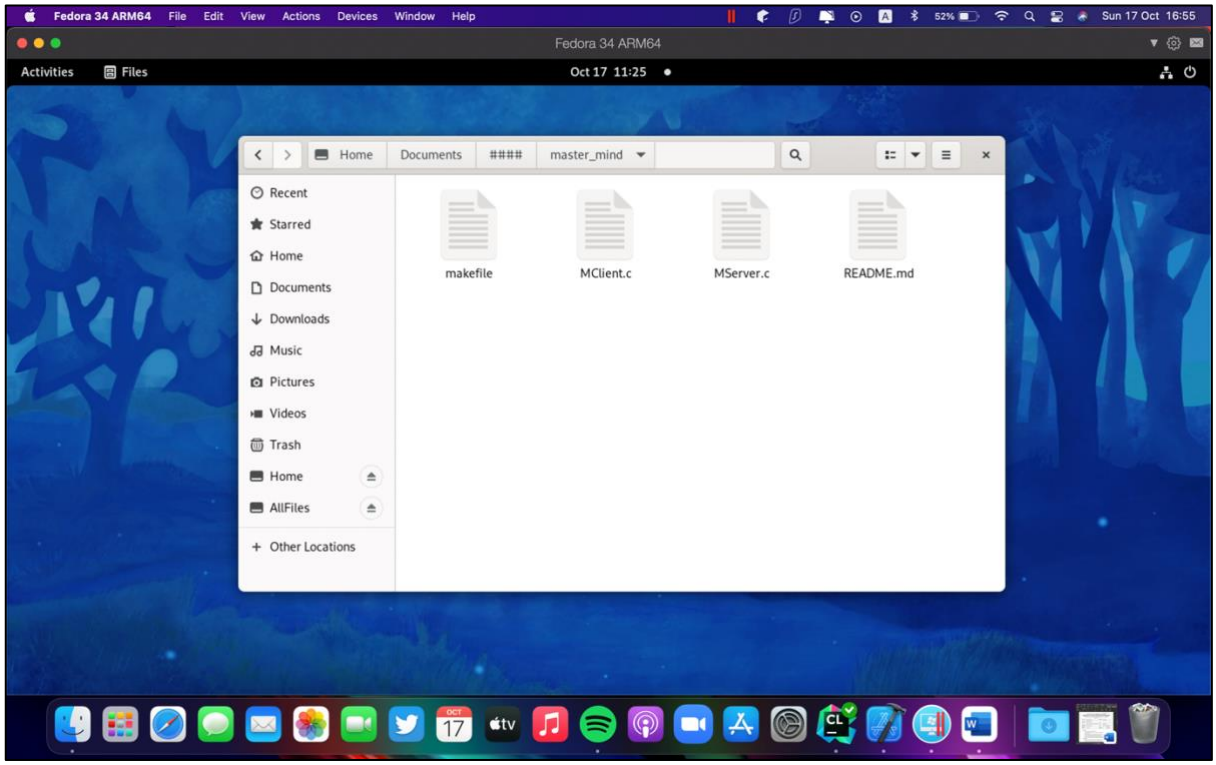

Figure 3.1: master_mind folder

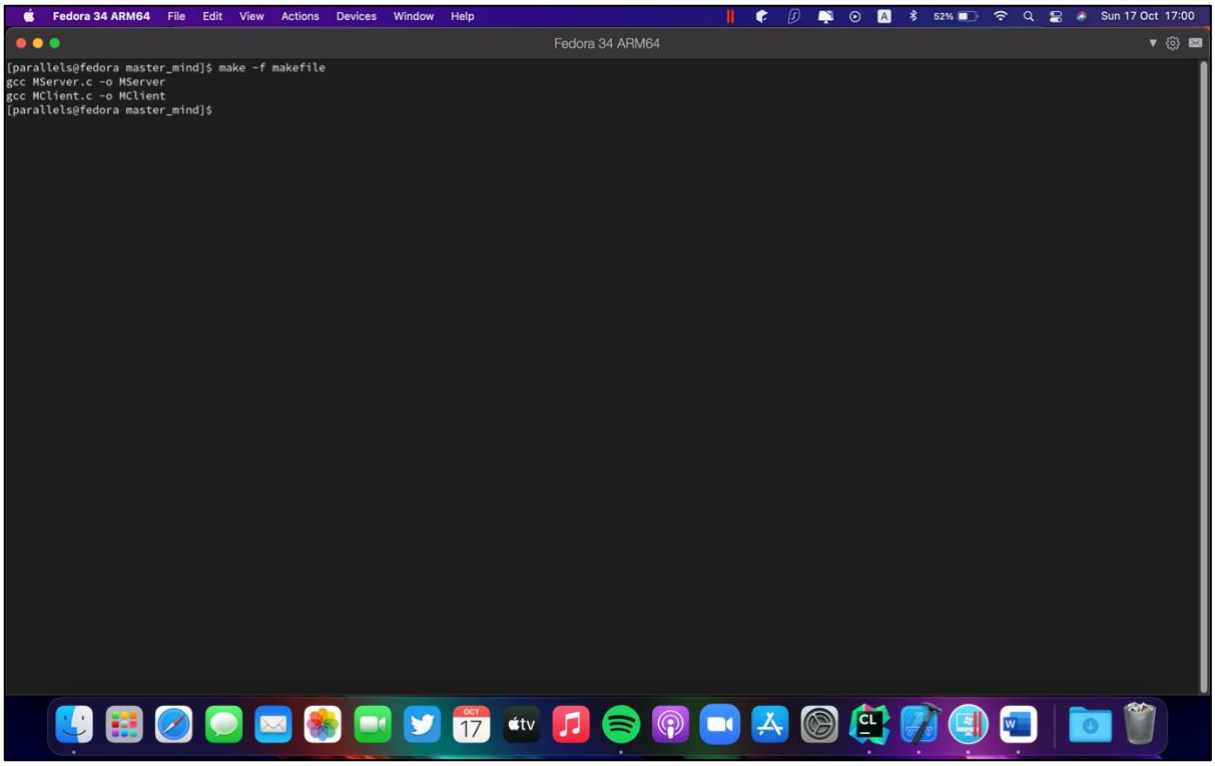

Figure 3.2: Runing the makefile

After run the above command it create two object files in the directory (master_mind folder). Then type the following commands on another terminal tab to run the above created object files.

```
./MServer 127.0.0.1 1234
./MClient 127.0.0.1 1234
```

Note: In here to avoid some errros and to test the programme, I used the above command type. In the assignment it say to run the command "./MServer 10 5 20 5 100 8". So in the code I gave the fixed values to the M = 10, N = 5, G = 20, T = 5 minutes, max_wait_game = 100 seconds, and max_clients = 8. Due to the limited time period I was unable to correct that.

After running the above commands in separate terminal tabs it gives the outputs as following.

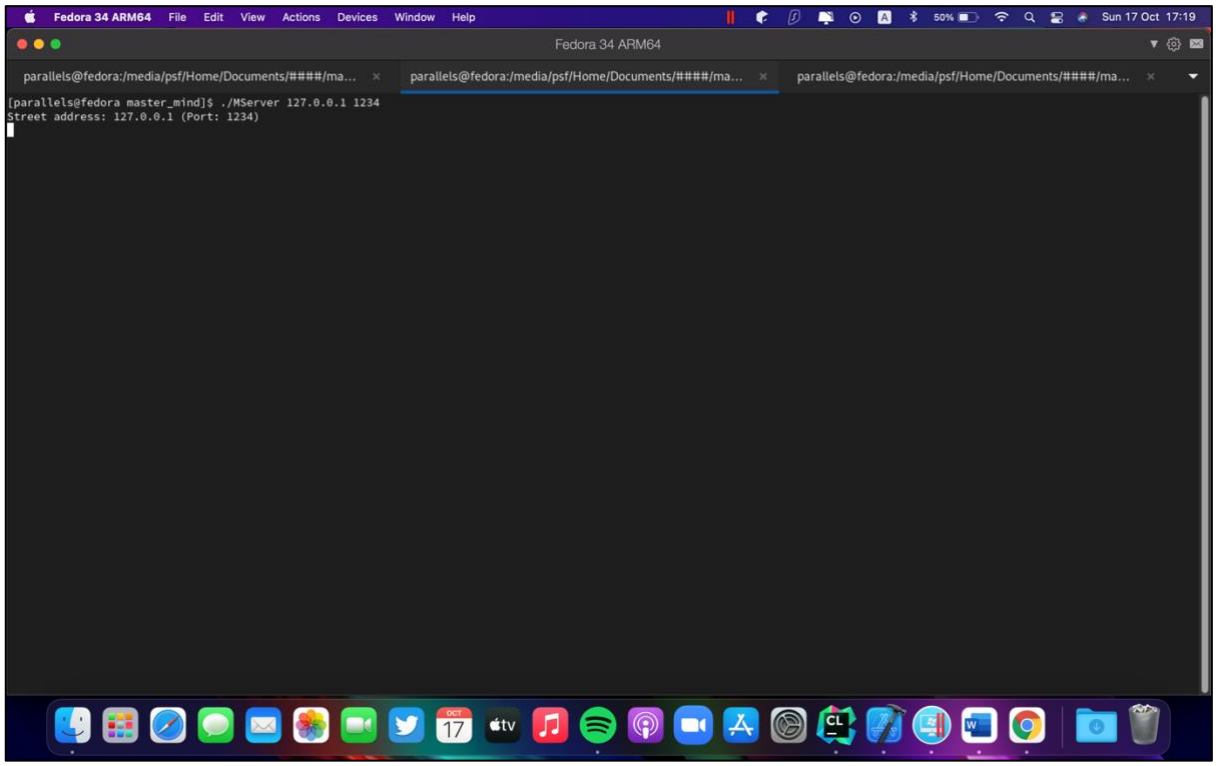

Figure 3.3: Running the MServer

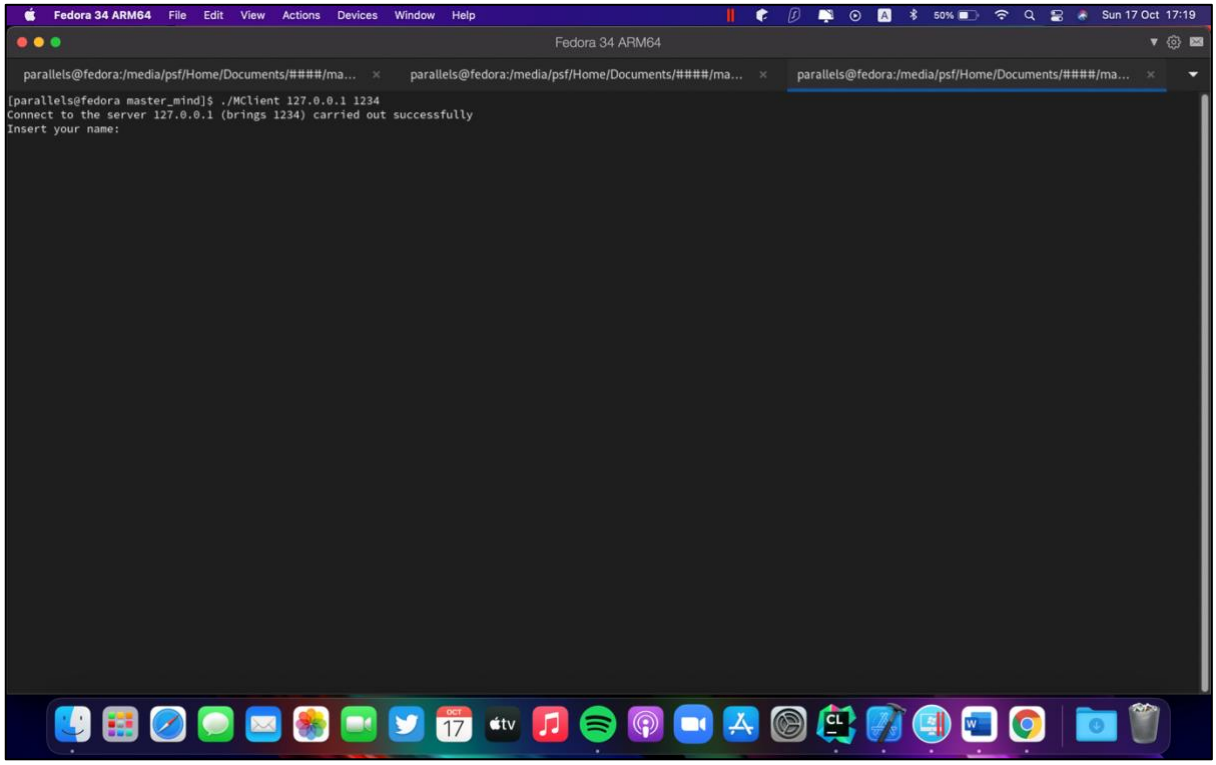

Figure 3.4: Running the MClient

After running the MClient it connects to the MServer and it asks for the player name (figure 3.4). Then it asks for the player port number (figure 3.5). Player should give the valid port number (port < 1024 || port > 65535) to continue to the game.

After entering the valid port number it shows the welcome interface of the game as following (figure 3.5).

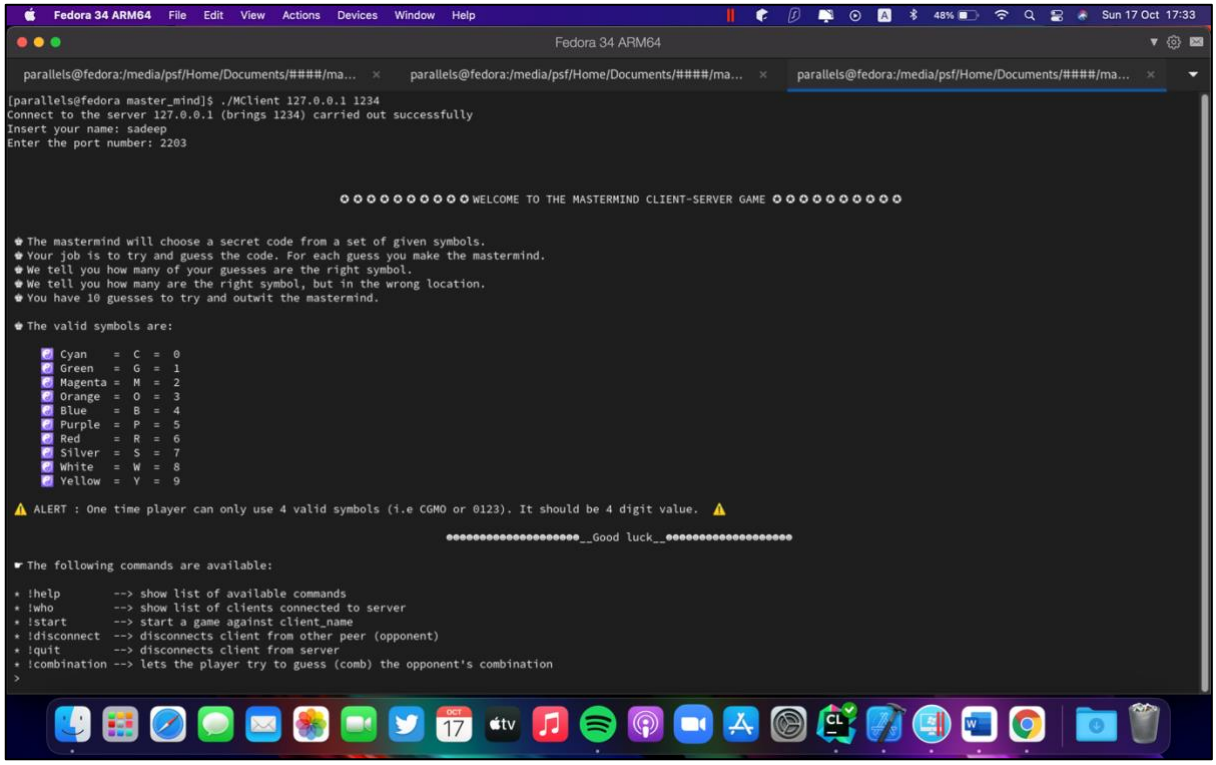

Figure 3.5: Welcome Interface

In this interface player can see the list of valid commands and valid symbols that can use in the game.

Note: In here to compare the players inputs with other players uses the numeric value system. In the assinggment it says to use the Blue = B, Cyan = C, Green = G, Magenta = M, Orange = O, Purple = P, Red = R, Silver = S, White = W, Yellow = Y. But in this code I used 0-9 values and these values substitute to the above alphabetic values.

In the welcome interface player can see the list of commands and using these command player can get help, stat the game, see whose playing the game, see the other playrs gusses and quit the game.

So in here ten players can play this game at once. Every player have to give the command "./MClient 127.0.0.1 1234" to join the game (Figure 3.6). Also they all have to give the valid port numbers. After connecting to the server they all can see the welcome interface.

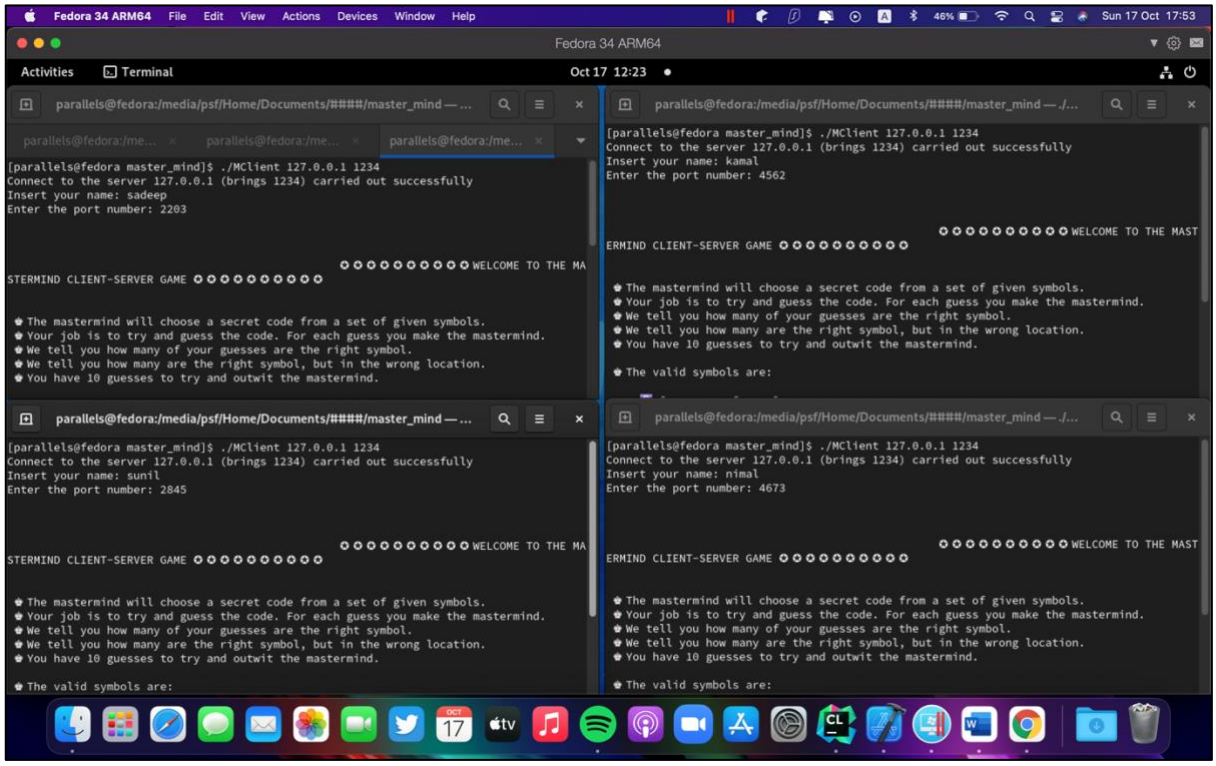

Figure 3.6: Join the Players

Also a player can see the who are the other players that available to play the game. The players who are not in a game have "FREE" tag after their name and the players who are currently in a game have "BUSY" tag agter their name (Figure 3.7). To see the status user only have to type the command "!who" in the terminal(figure 3.7).

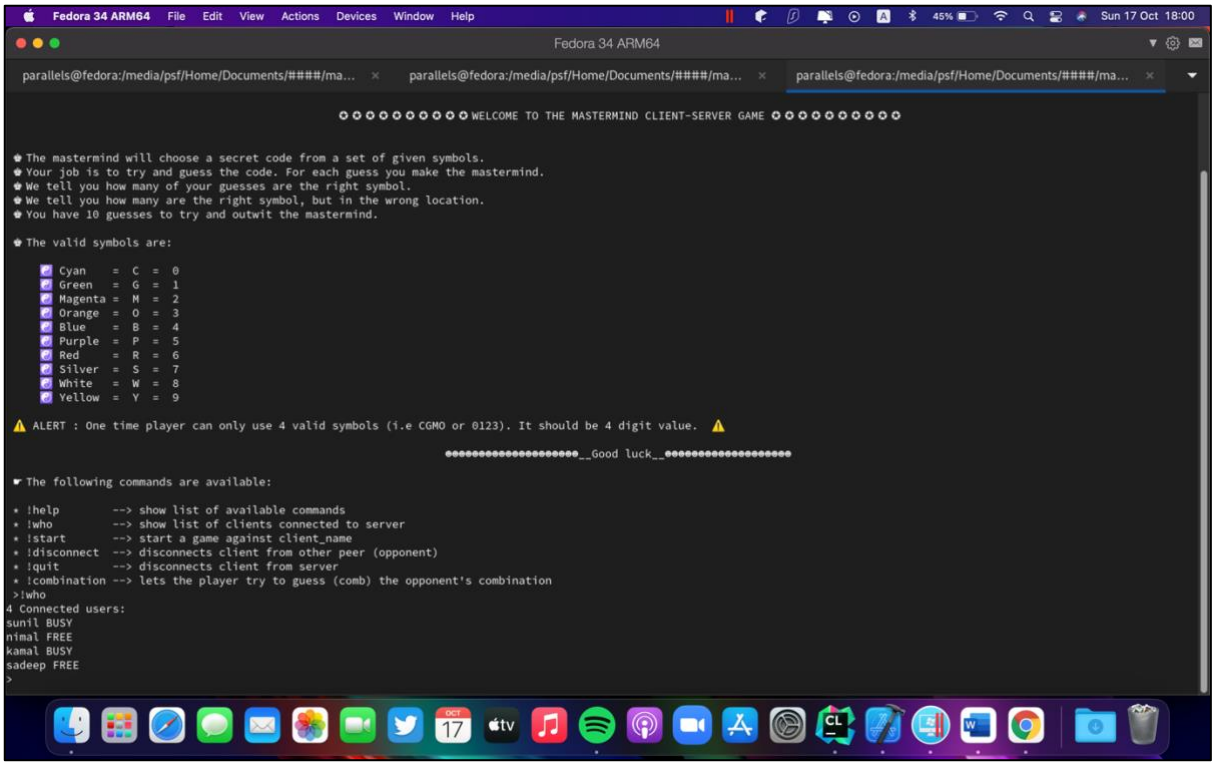

Figure 3.7: Player Status

In here (figure 3.7) Sunil and Kamal are already in a game so their names shown with the BUSY tag.

So to start the game player have to give the command “!start” and after that type the players name that you wish to play with (figure 3.8). After that it sends a invitation to the that player. If the player wish to play the game him/her can accept the invitation (figure 3.9).

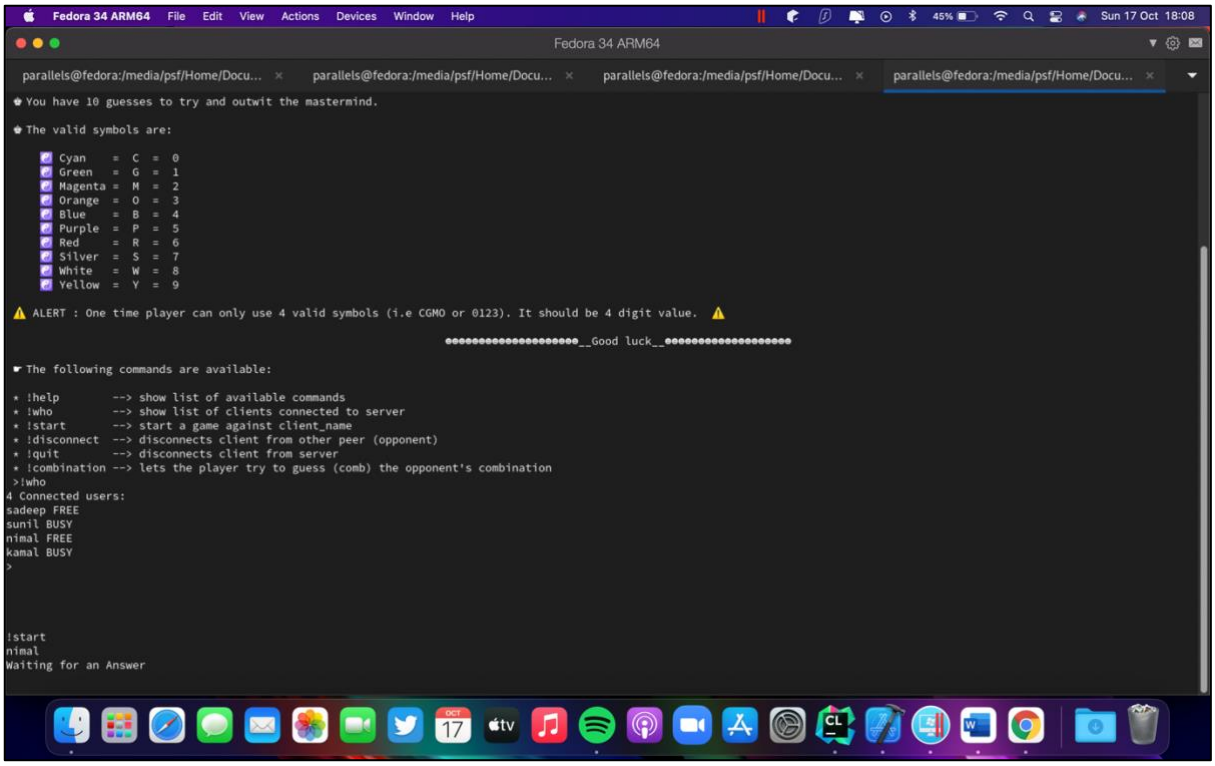

Figure 3.8: Send request to the player

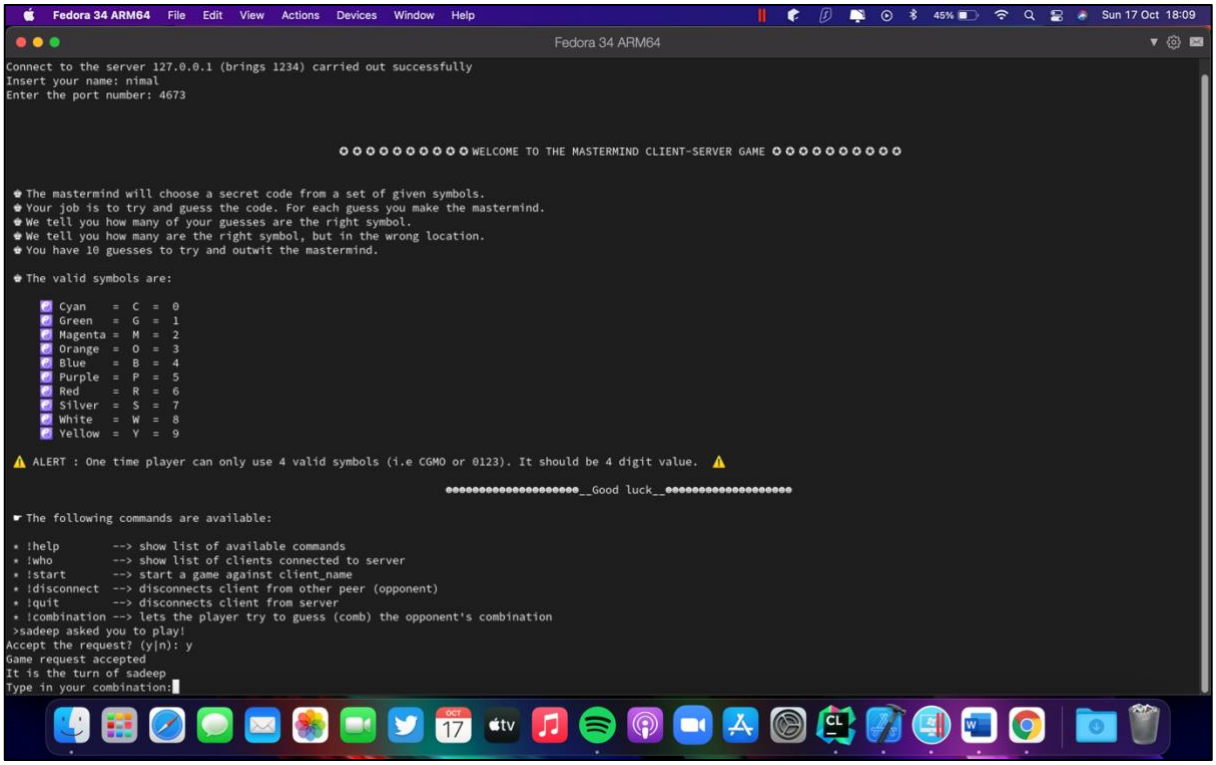

Figure 3.9: View the send request and accept or deny

If the other player accepts the invitation player can start the game (figure 3.9). Now player can enter their guess.

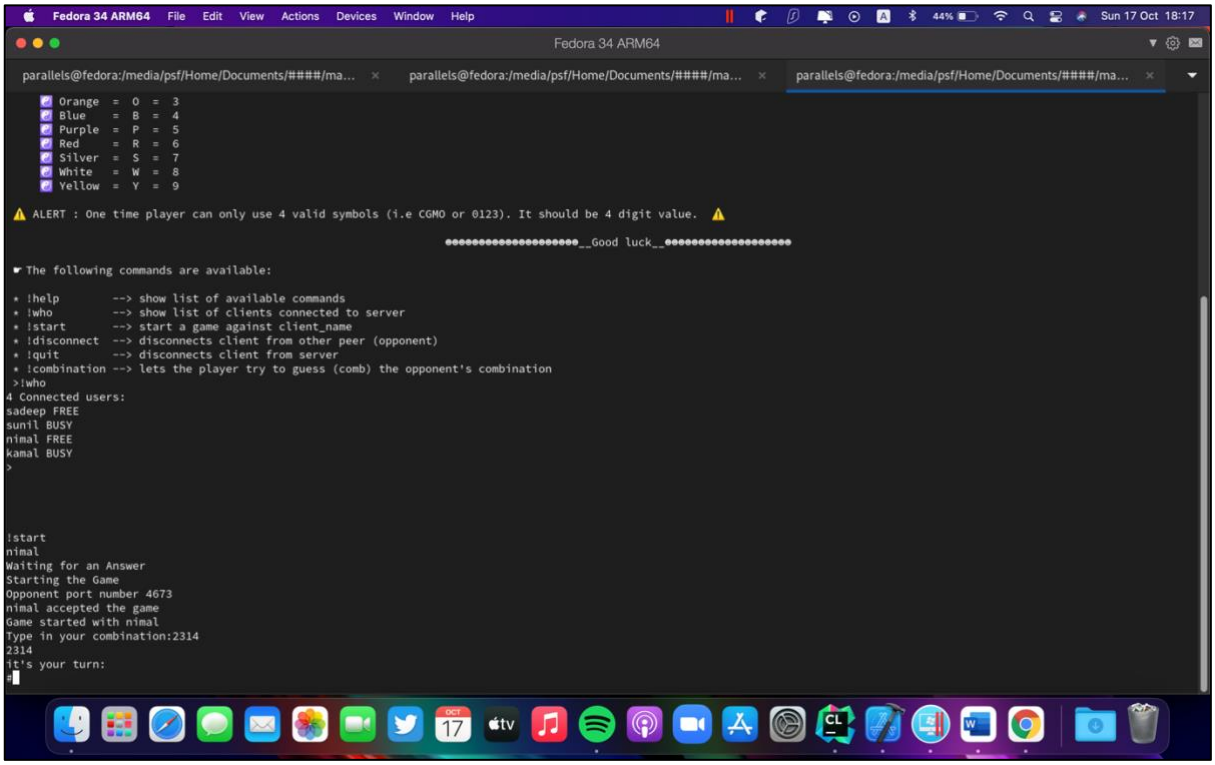

Figure 3.10: Enter the guess

After the first player enter the code you can enter the your guess. To see the how mush correctly guessed you can type the command “!combination” and give the guess you enterd previously. Then it shows the how much of your guess is correct (figure 3.11).

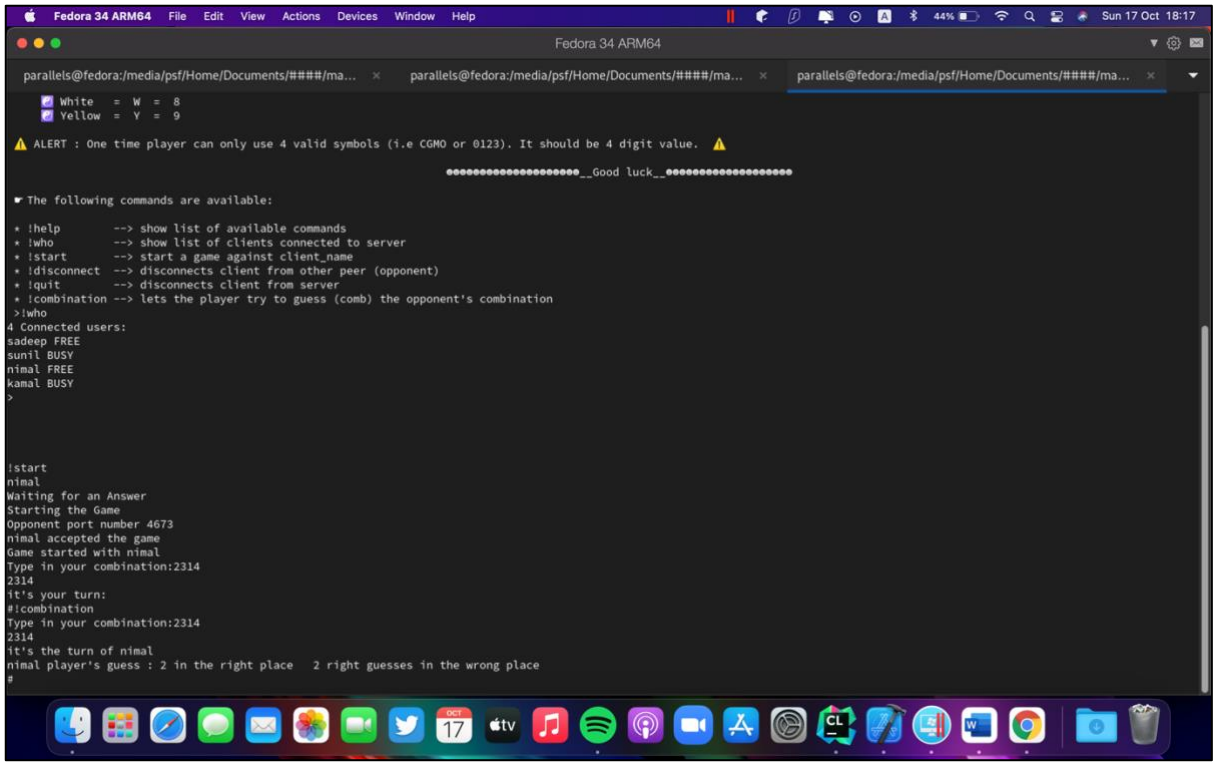

Figure 3.11: See opponent guess

Like this you can continue your guess until you win. If you unable to guess the opponent guess within the given time period opponent can win the game (figure 3.12). Maximum time limit to guess is 60 seconds.

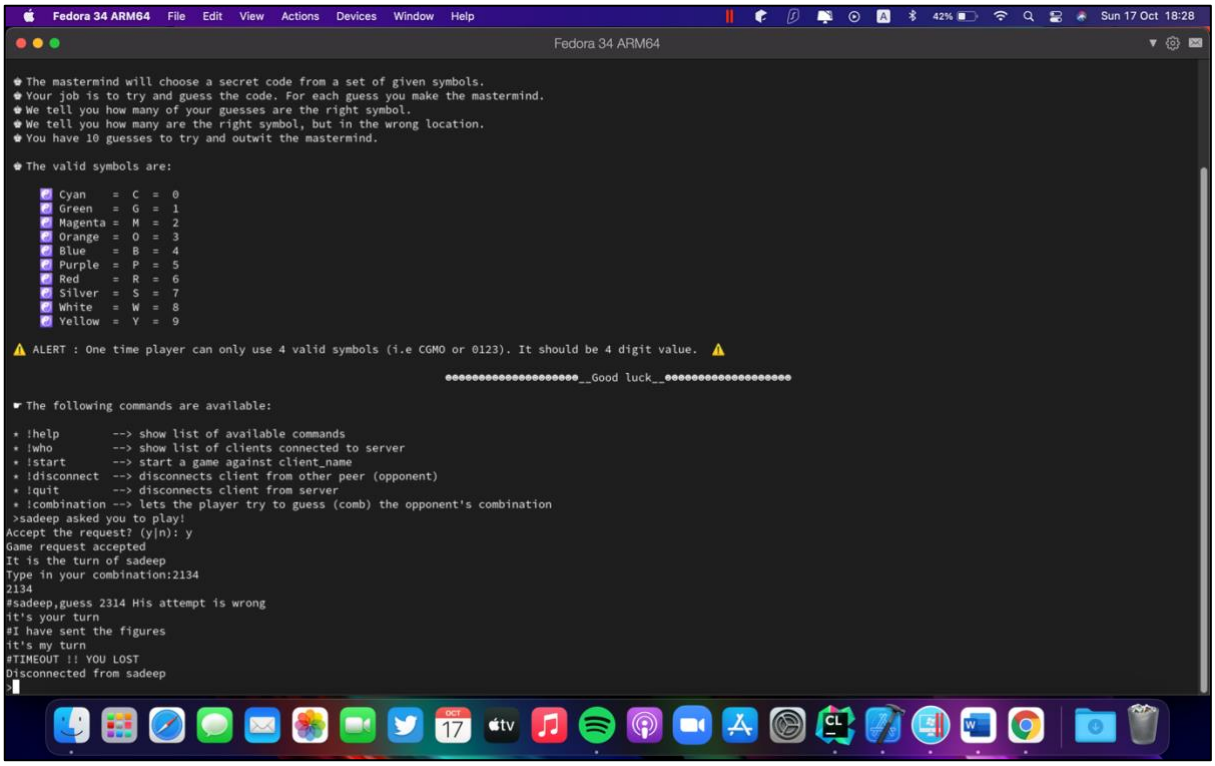

Figure 3.12: Player timeout lost

After the player unable to guess within given time period you win the game (figure 3.13).

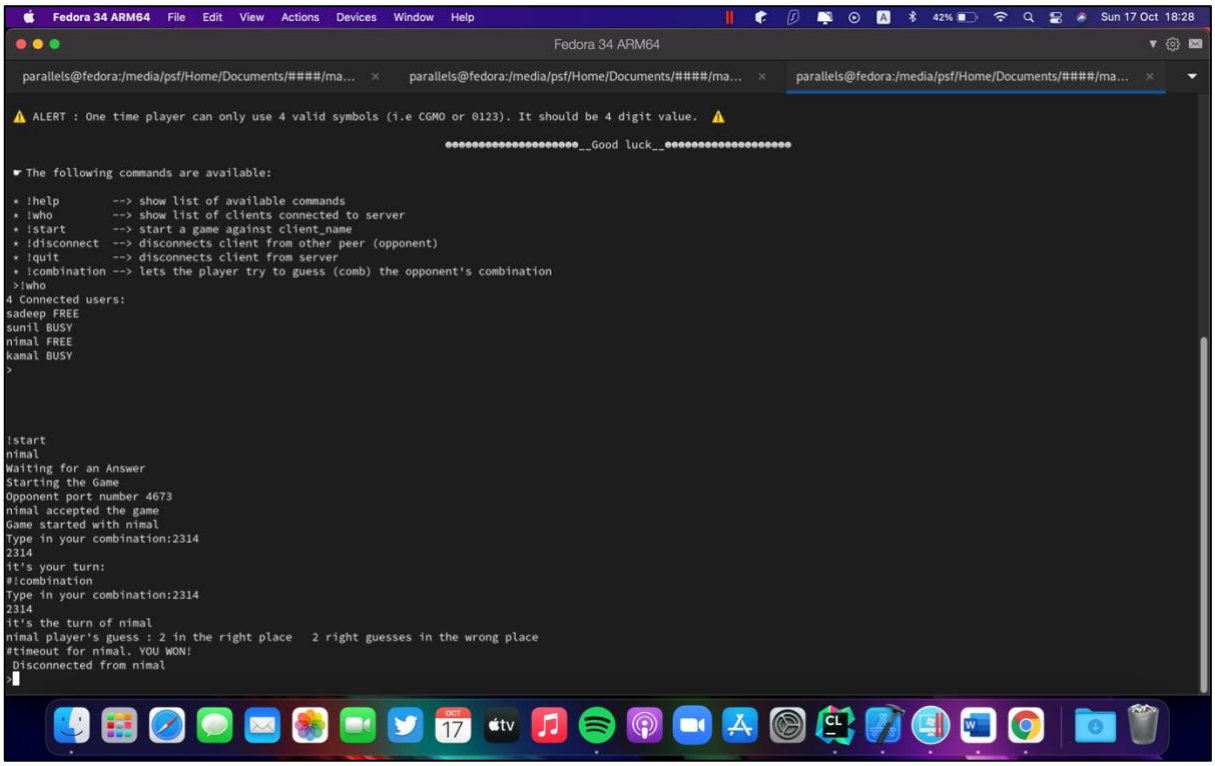

Figure 3.13: Player timeout win

Also if the player want to quit the game he can use the command “!quit”. If you quit the game in the middle of the game opponent can win.

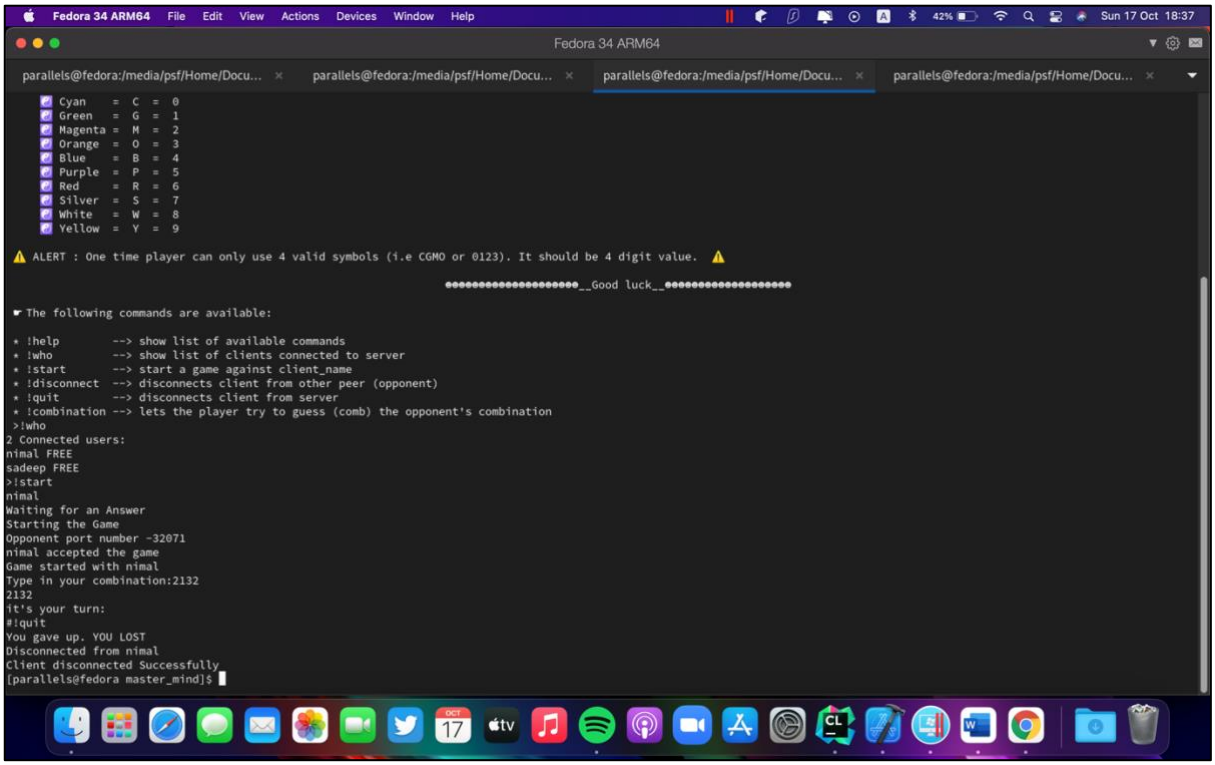

Figure 3.14: Player quit the game

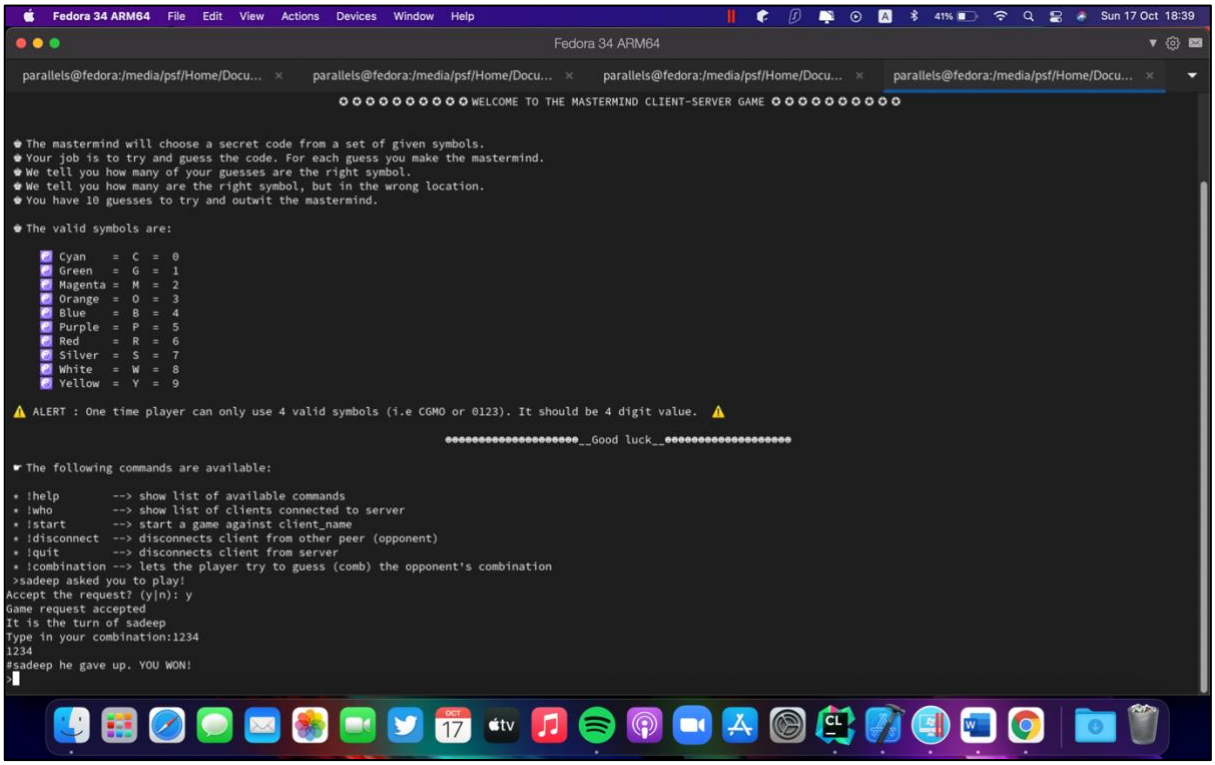

Figure 3.15: Opponent win

4. Server Programme

4.1 Shared data structures used in the server (MServer)

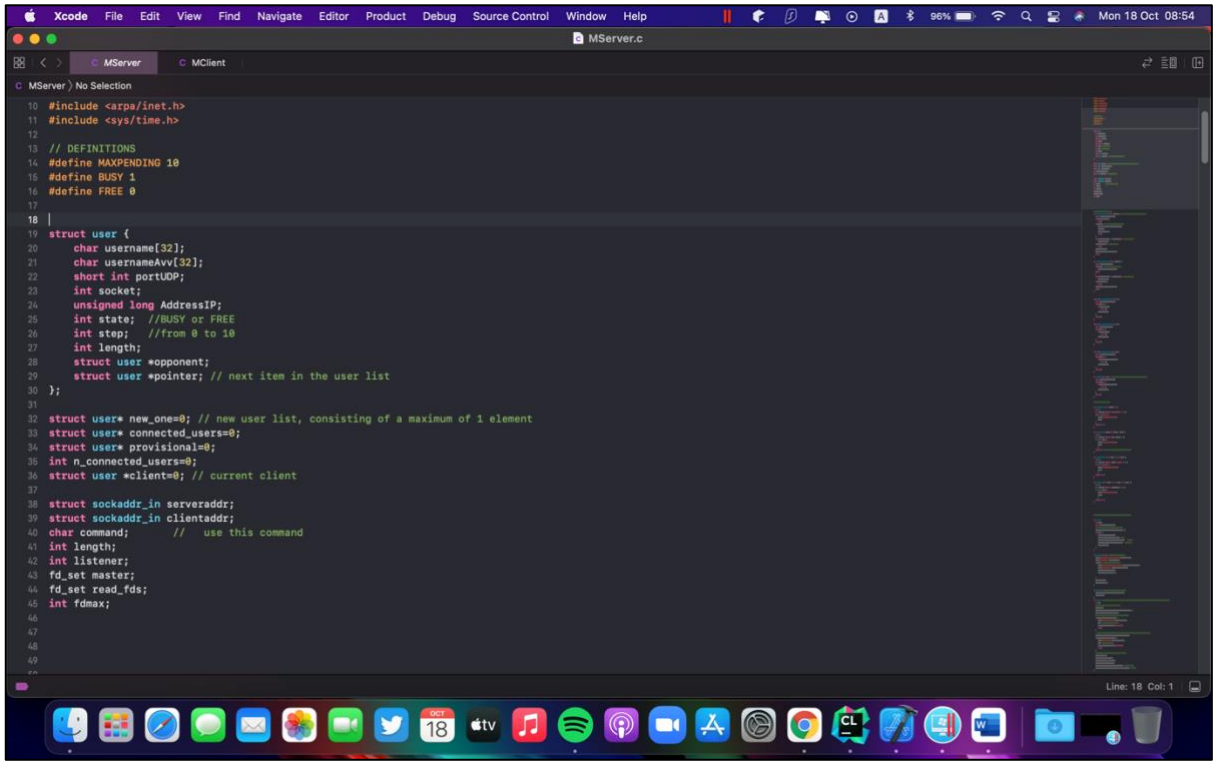

Figure 4.1.1: Shared Data Structures in MServer

In here there are many data structures used. But these are the (figure 4.1.1) shared data structures that shares resources with the client (MClient) programme. Below you can see the detailed discussion on all the mentioned structures.

```
struct user {
        char username[32];
        char usernameAvv[32];
    short int portUDP;
        int socket;
        unsigned long AddressIP;
        int state;          //BUSY or FREE
        int step;//from 0 to 10
        int length;
        struct user *opponent;
```

```
        struct user *pointer; // next item in the user list
};
```

The above structure used to collect the user information. These information saves in the memory locations and these saved information used as a input in the functions. After a successful connection between client and server this structures in the saver connects with the structures in the client and they share the information.

```
struct user* new_one=0; // new user list, consisting of a maximum of 1 element
struct user* connected_users=0;
struct user* provisional=0;
int n_connected_users=0;
struct user *client=0; // current client
```

These set of structures in the server programme used to get a new user connection, count the connected users and get client details. As told earlier these stuctures are getting the data from client programme.

```
struct sockaddr_in serveraddr;
struct sockaddr_in clientaddr;
```

The above two structures used to get the server address and the client address. The "struct sockaddr_in serveraddr;" structure also used in the client programme to get the server address and to connect with the server.

After getting the data to these structures, they passes their values to the functions that created in the server programme. They are shown in the below.

```
void remove_item_from_list(struct user* elem)
```

```
void remove_from_provisional(struct user* elem)
```

The above two function getting the struct user as a input. These functions are use to remove a player from the list and remove temporary created player from the list.

```
struct user* find_conn_from_socket(int sd) {
        struct user* temp=connected_users;
        while(temp!=NULL) {
          if(temp->socket==sd)
                  return temp;
          temp=temp->pointer;
        }
        return NULL;}
```

In here using the user structure it create function name "find_conn_from_socket" and it substitute the connected_users to temp variable that crated from the user structure. This function is used to pass the socket and look for the list of users that connected to the game.

```
struct user* find_prov_from_socket(int sd) {
        struct user* temp=provisional;
        while(temp!=NULL) {
          if(temp->socket==sd)
                  return temp;
          temp=temp->pointer;
        }
        return NULL;
}
```

Using the above structure it searches for the temporary created sockets and players. In here the temporary means the players that join the game and quit the game immediately. The list of those players are also keep the server.

```
struct user* find_from_username(char* name) {
        struct user* temp=connected_users;
        while(temp!=NULL) {
          if(strcmp(temp->username,name)==0)
                  return temp;
          temp=temp->pointer;
        }
        return NULL;
}
```

Above structure use to send the recipient name and request information (username, port number) on the client who wants to play with him.

5. Assumptions,Limitations and Errors

5.1 Assumptions

In here to avoid some errros and to test the programme, I used the below command type. In the assignment it say to run the command "./MServer 10 5 20 5 100 8".

```
./MServer [loop back IP] [Port number]
```

```
./MServer 127.0.0.1 1234
```

```
./MClient 127.0.0.1 1234
```

So in the code I gave the fixed values to the M = 10, N = 5, G = 20, T = 5 minutes, max_wait_game = 100 seconds, and max_clients = 8. Due to the limited time period I was unable to correct that. [Mentioned in page 28]

In here to compare the players inputs with other players this programme uses the numeric value system. In the assinggment it says to use the Blue = B, Cyan = C, Green = G, Magenta = M, Orange = O, Purple = P, Red = R, Silver = S, White = W, Yellow = Y. But in this code I used 0-9 values and these values substitute to the above alphabetic values. [Mentioned in page 30]

```
Cyan      = C  = 0
Green     = G  = 1
Magenta   = M  = 2
Orange    = O  = 3
Blue      = B  = 4
Purple    = P  = 5
Red       = R  = 6
Silver    = S  = 7
White     = W  = 8
Yellow    = Y  = 9
```

Content of the HowToPlay file built in to the MClient programme. So there is a no file in the directory.

5.2 Limitations

In this programme the “Stat” command is not working so the players can not recive the previous game details. It means the S table is not working. They are the total number of players that have played the game, total number of players that have successfully guessed secret code, fastest time a player has guessed secret code, the smallest number of guesses a player has guessed secret code, the total number of players that fail to correctly guess secret code due to time limit and the total number of players that fail to correctly guess the secret code due to guess limit.

5.3 Errors

After running the programe several times there are no erros other than the limitations. So the code and programme run smoothly and the you can see the outputs in the below figures.

6. Sample Inputs and Outputs of the Programs

6.1 Inputs and Outputs

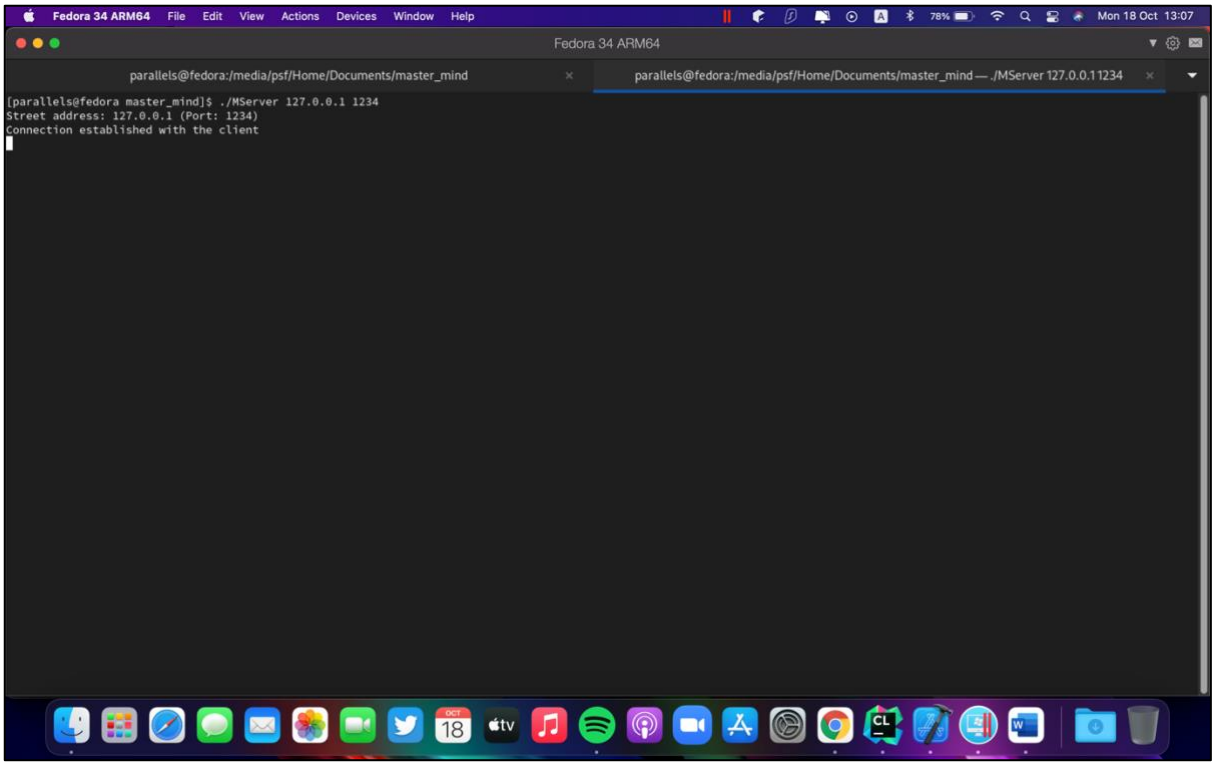

Figure 6.1.1: MServer Inputs

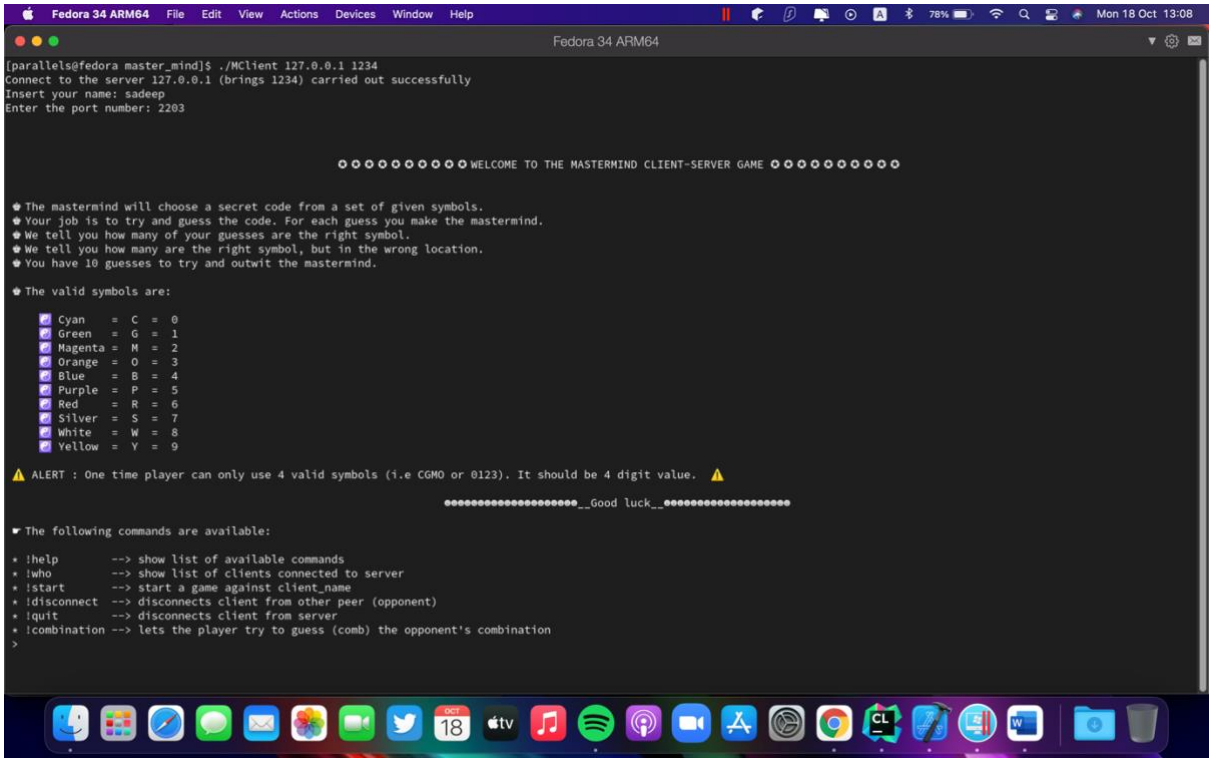

Figure 6.1.2: MClient Inputs

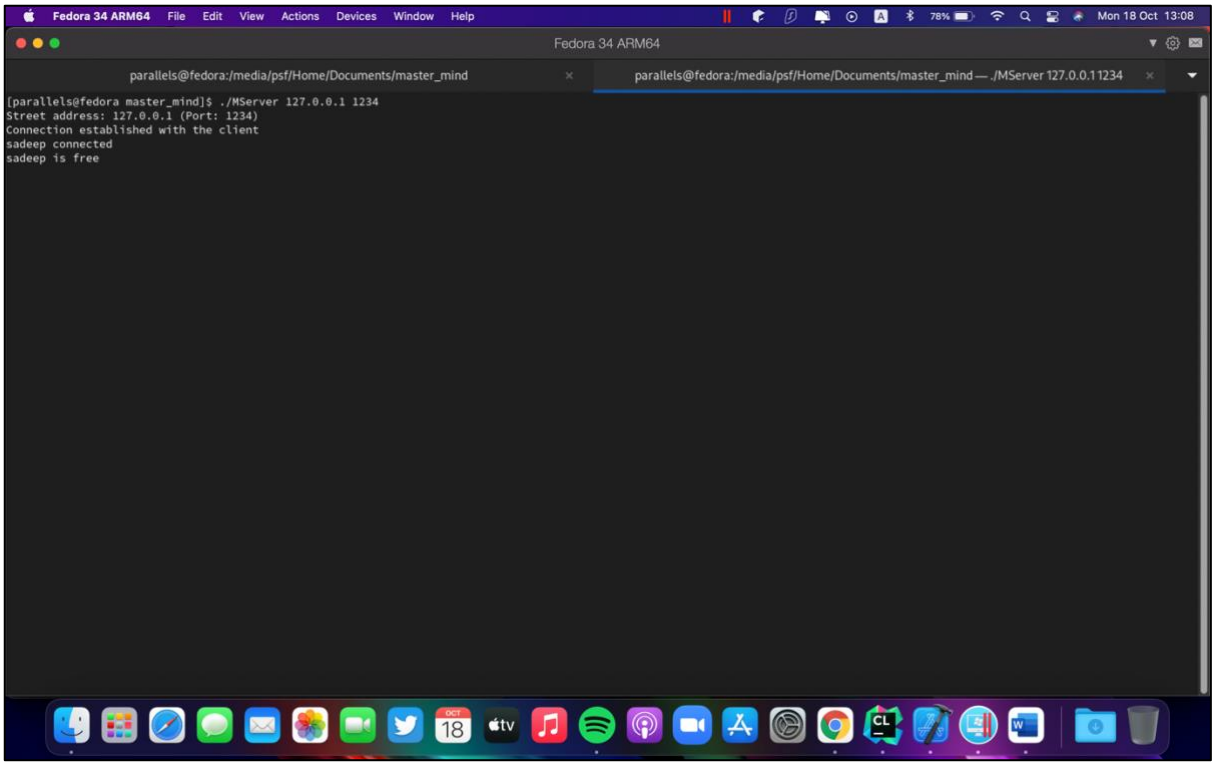

Figure 6.1.3: MServer view after Client connection

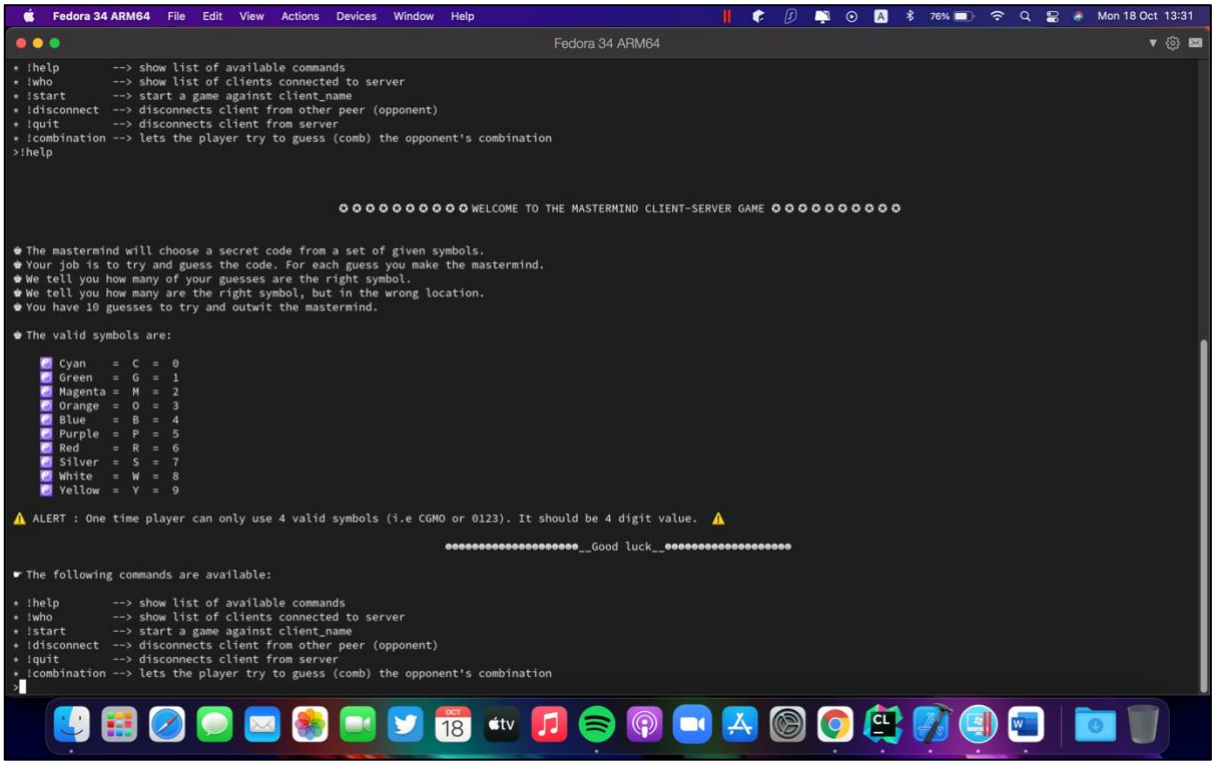

Figure 6.1.4: “help” command input and its output

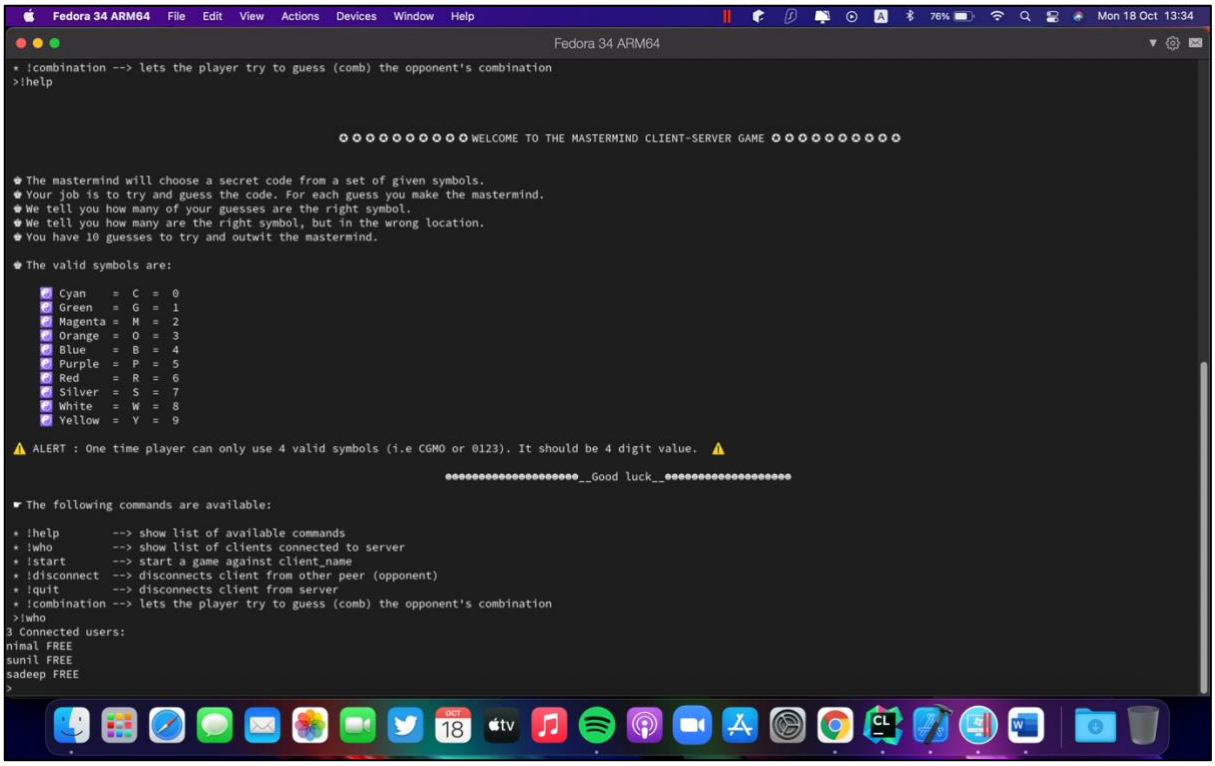

Figure 6.1.5: “who” command input and output (see the available players)

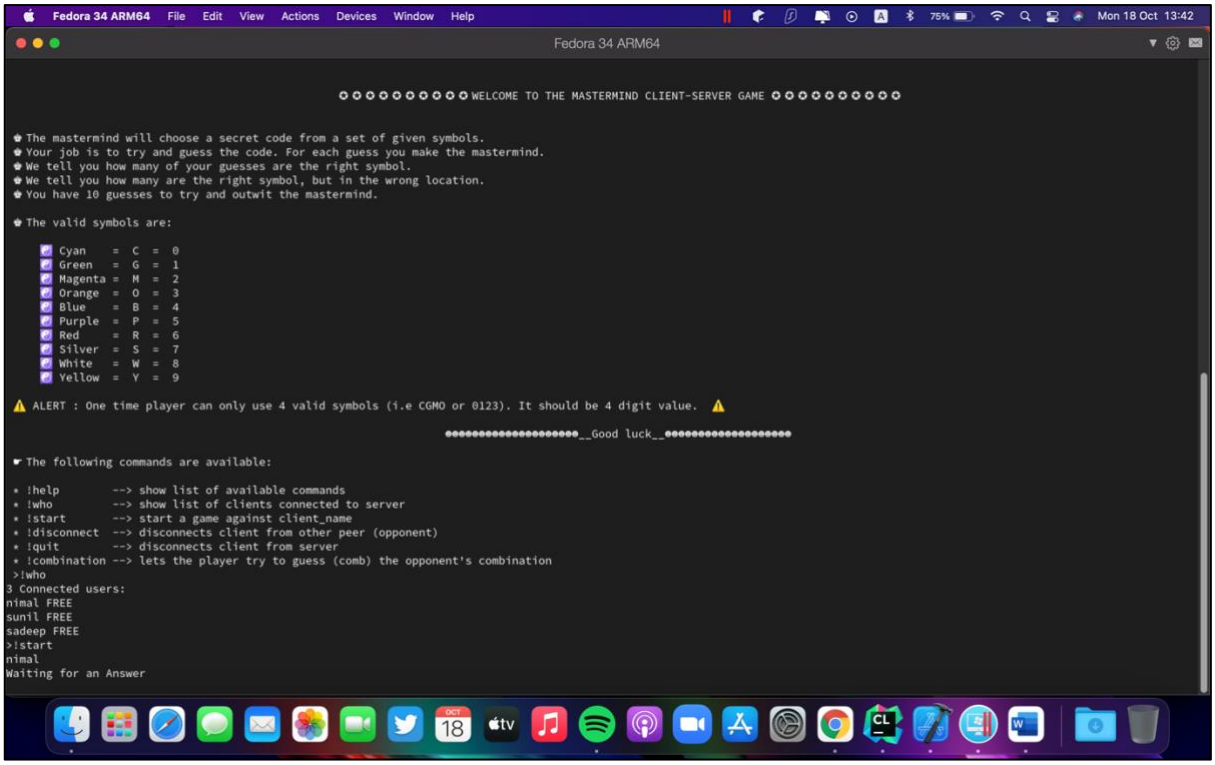

Figure 6.1.6: “start” command input in player sadeep

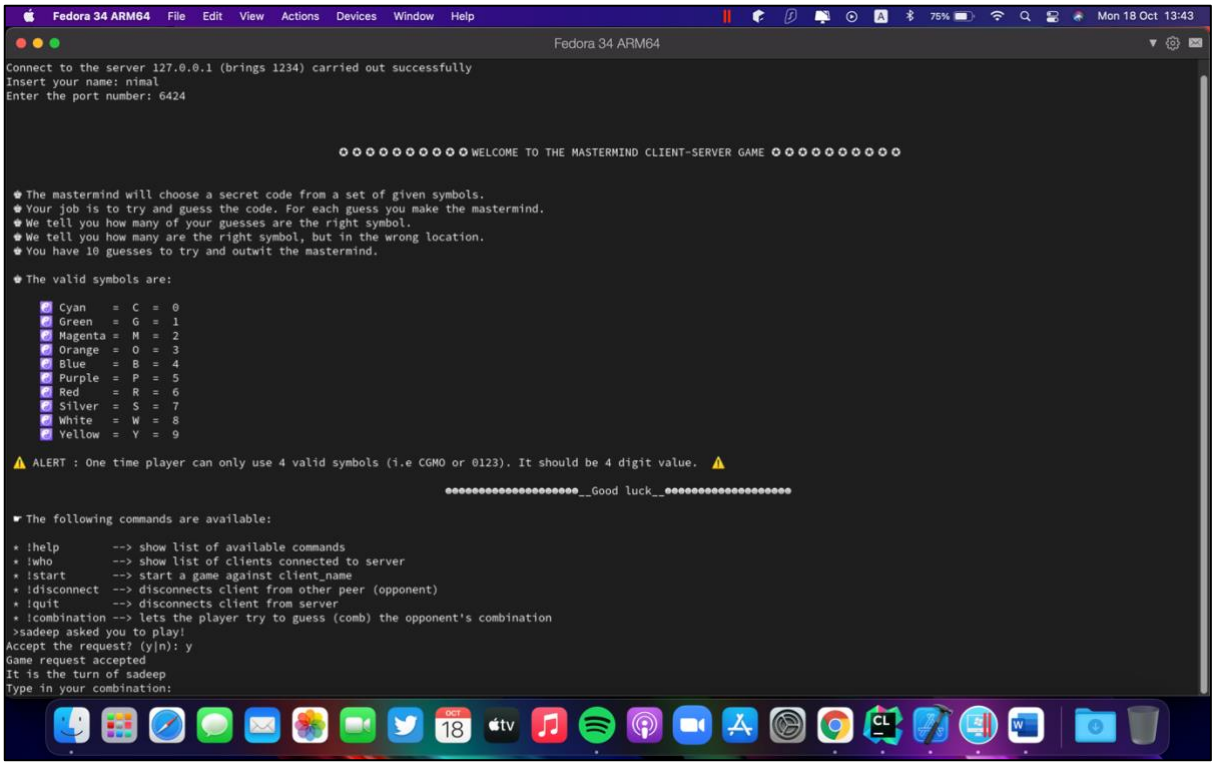

Figure 6.1.7: received game request from player sadeep

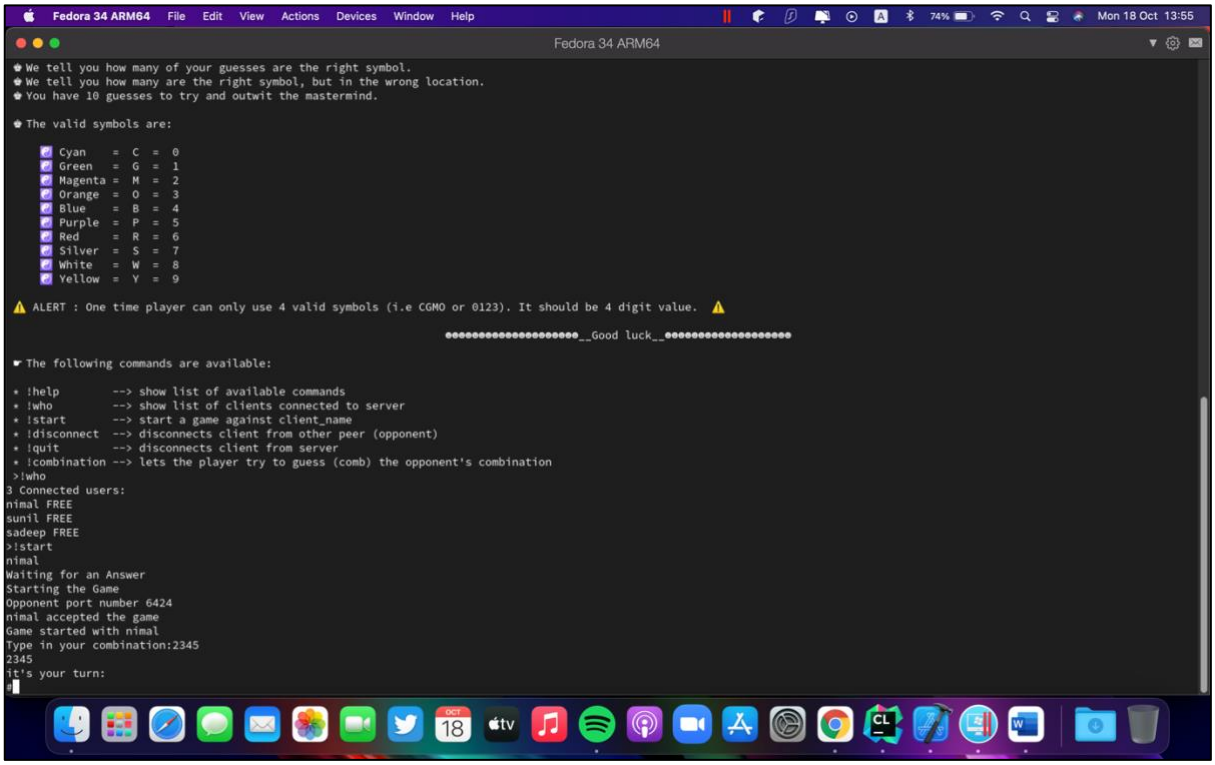

Figure 6.1.8: Game started with nimal

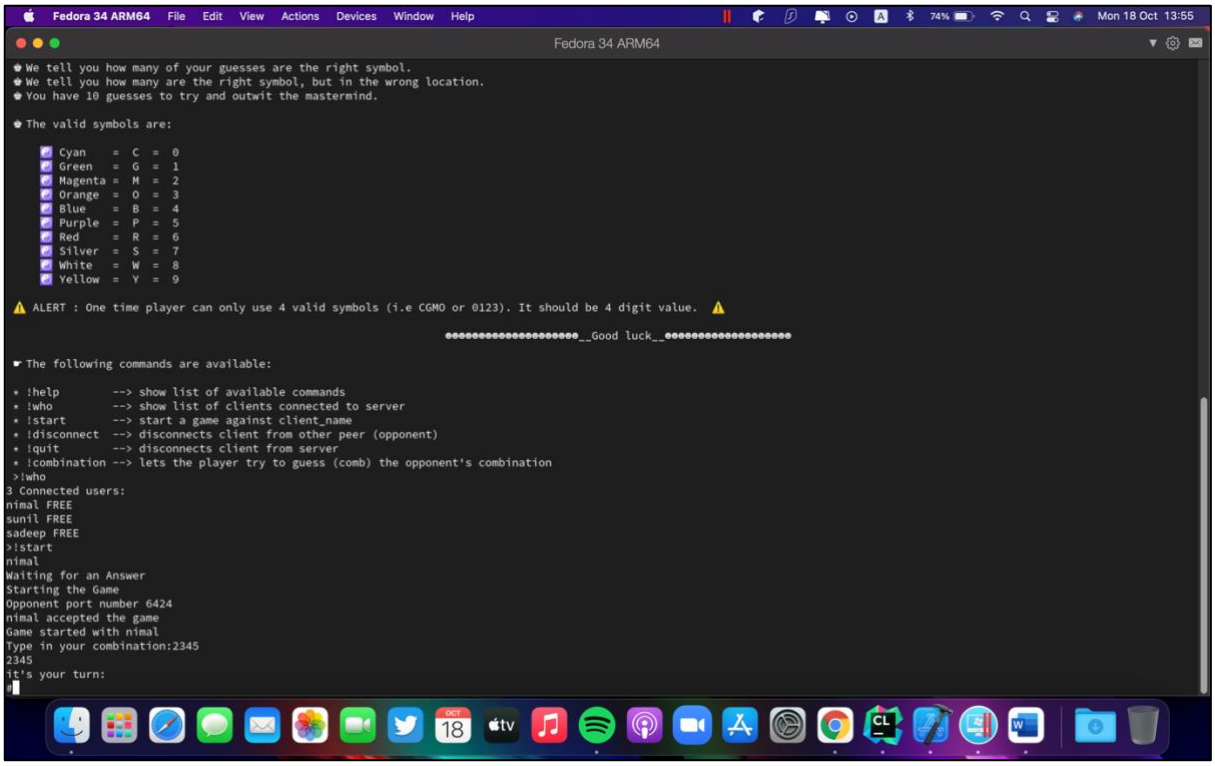

Figure 6.1.9: Input the guess (player sadeep)

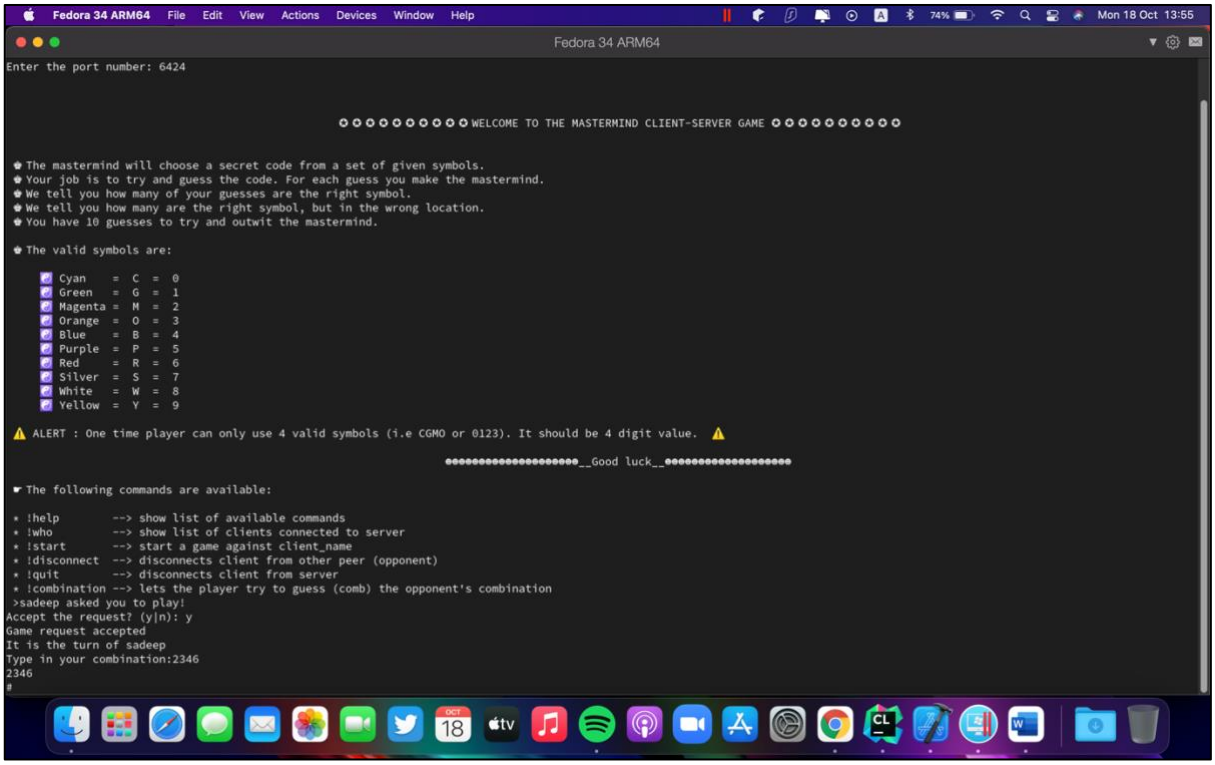

Figure 6.1.10: Input the guess (Player nimal)

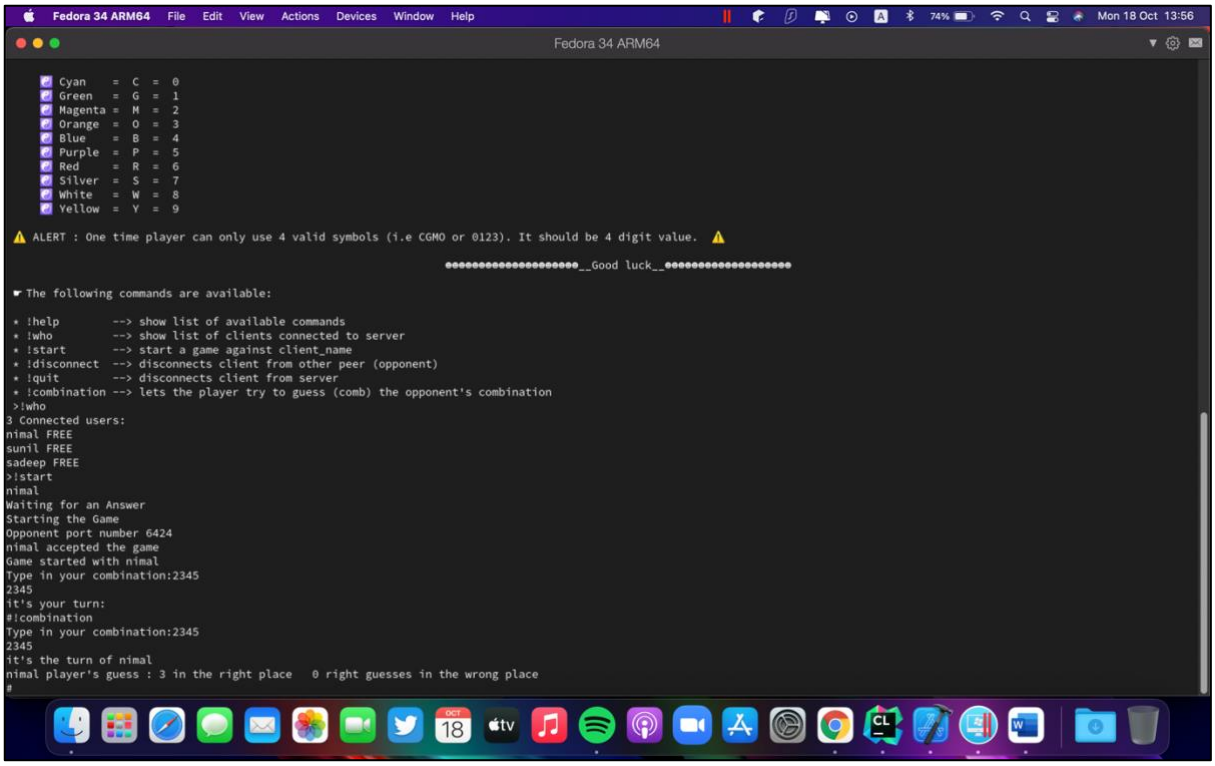

Figure 6.1.11: "!combination" command in player sadeep

Using this command you can see how many guesses in the right place and how many are correct. Like this you can continue this game until you win.

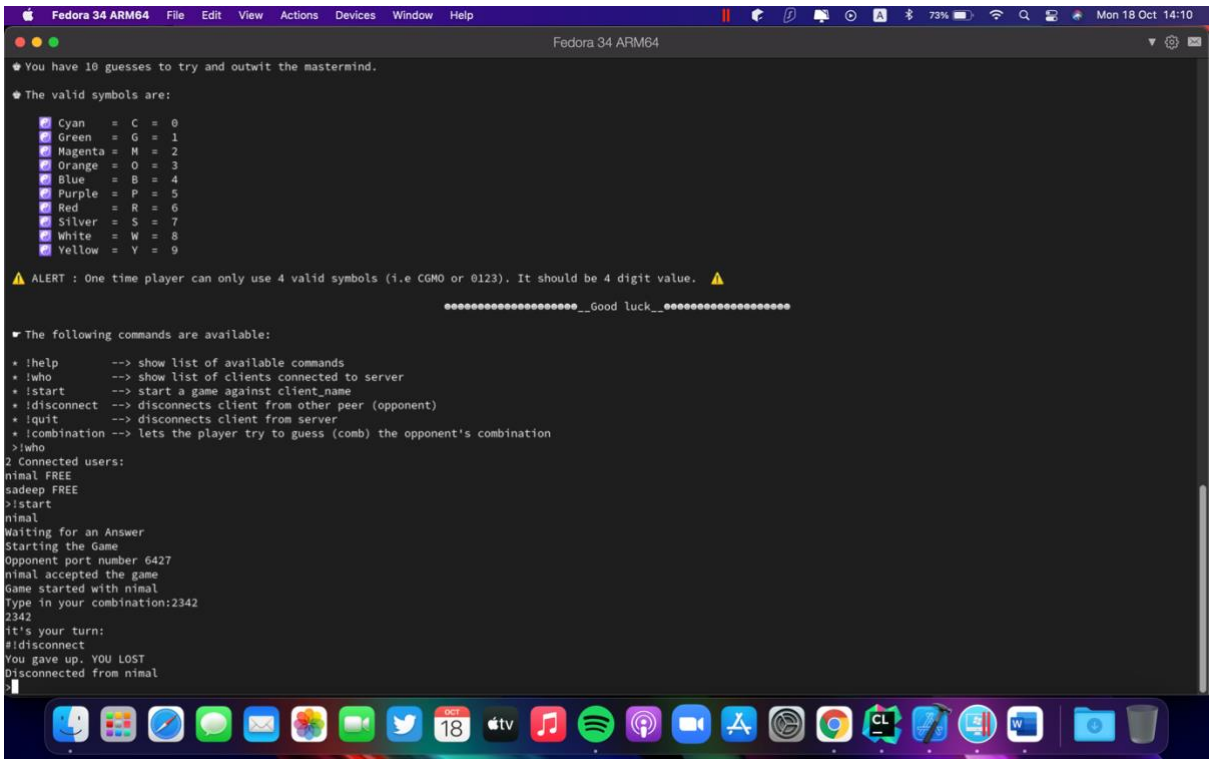

Figure 6.1.12: "disconnect" command in player sadeep

After using the ‘”disconnect” command player disconnects from the other player and you loose the game.

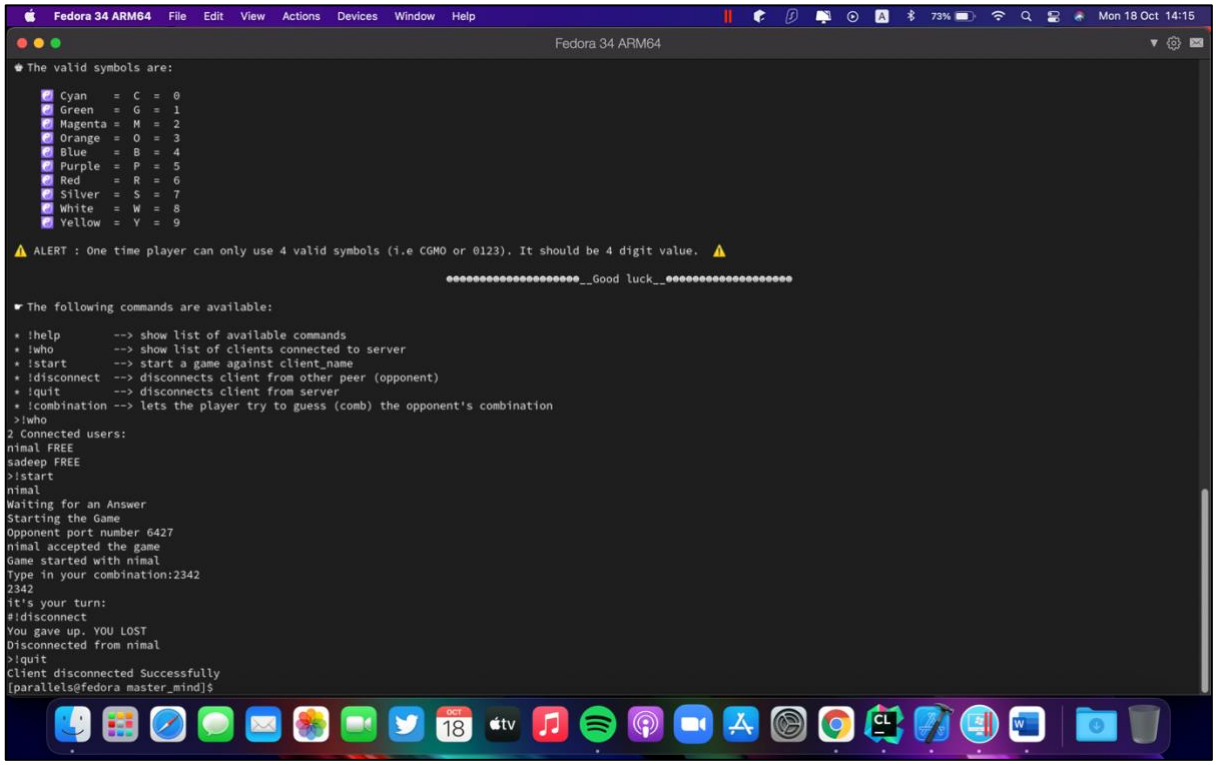

Figure 6.1.13: “quit” command

After using the quit command you disconnects from the server and you no longer can play the game.

7. References

- CODE PROJECT. How to send a record to server using socket,i am not getting data at server end [online] Available https://www.codeproject.com/Questions/1011568/How-to-send-a-record-to-server-using-socket [accessed date 07/10/2021]

- Check Socket Connection State [online] Available https://coderanch.com/t/205451/java/Check-Socket-Connection-State [accessed date 10/10/2021]