BATTLESHIP

A COURSE PROJECT REPORT

By

NIHAL SHAH PAREKKATTIL (RA1911003010868)
SURAJ ARS (RA1911003010872)
YASH P SHAH (RA1911003010881)

Under the guidance of

DR. V. DEEBAN CHAKRAVARTHY

In partial fulfilment for the Course

of

18CSC302J - COMPUTER NETWORKS

in

Computer Science & Engineering.

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District

NOVEMBER 2021

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

It is to certify that this project report BATTLESHIP is the bonafide work of NIHAL SHAH PAREKKATTIL(RA1911003010868) SURAJ ARS (RA1911003010872) YASH P SHAH (RA1911003010881), who carried out the project work under mysupervision.

SIGNATURE	**SIGNATURE**
Dr T R Saravanan	**Dr.E. Sasikala,**
Assistant Professor	**Course Coordinator**
CSE	**Associate Professor,**
SRM Institute of Science and Technology	**Data Science and Business Systems**
Potheri, SRM Nagar, Kattankulathur,	SRM Institute of Science and Technology
Tamil Nadu 603203	Potheri, SRM Nagar, Kattankulathur,
	Tamil Nadu 603203

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement. We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to the Chairperson, School of Computing **Dr. Revathi Venkataraman**, for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr.M.LAKSHMI, Professor and Head, Data Science and Business Systems and Course Coordinator Dr.E. Sasikala, Associate Professor, Data Science and Business Systems** for their constant encouragement and support.

We are highly thankful to our Course project Internal guide **Dr P MADHAVAN , Assistant Professor , Computer Science and Engineering**, for **his** assistance, timely suggestion and guidance throughout the duration of this course project.

We extend our gratitude to the **Students, Dr.B.Amutha ma'am (HOD)** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

1. ABSTRACT

The Project is an implementation of the classic arcade Battleship game on a multi-client supporting server platform. Battleship is a very popular strategy type guessing game for two players. It is played on a grid-usually on a paper grid or a board- on which each player's fleet of ships are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling "shots" at the other player's ships, and the objective of the game is to destroy the opposing player's fleet or all of the opponents ships.

Here sockets are used as a medium of communication. The architecture is based on message-passing paradigm, wherein a system serves as an intermediary among separate, independent processes. We employ the server as a daemon process, which connects two clients and set up the grid.

The main objective of this project to provide a throwback to the classic games that we used to play in childhood, and display the simplicity of porting the game into an online version, thereby connecting with more people.
Porting the game also is an educational necessity. Battleship game introduces school kids to coordinates and grids in a hands-on kind of way. It helps them understand the relationship between rows and columns, as well as the relationship between an actual object or a sequence of events and how they are represented on a graph. Battleship requires you to formulate a mental picture of your opponent's set up and to keep that picture in mind, making and remembering adjustments as the game unfolds and new information comes in. This helps kids develop their memory and strategy skills.
There are various iterations of the game, However, we have implemented the classic Battleship game.

The game is made using TCP/IP protocol and can be run in any computer device that has an IDE that can compile a C server client code

2. INTRODUCTION

The application we selected is a two-player board game Battleship. Battleship is a very popular strategy type guessing game for two players. It is played on a grid-usually on a paper grid or a board- on which each player's fleet of ships are marked. The locations of the fleets are concealed from the other player.

In Battleship, each player begins by placing its ships on a private board. Subsequently, the players take turns shooting cells of each other’s boards. After the opponent makes a shot on a player’s board, the player announces whether it was a “miss” (the shot cell was vacant) or a “hit” (the shot cell was part of an unspecified ship). When the last cell of a ship is hit, the player announces that this ship has been “sunk” (e.g., “you sank my battleship”). The first player to sink all its opponent’s ships is declared the winner.

Battleship is either a game of luck, or search algorithm optimization. However, deriving a optimum solution is difficult because the strategy of each players varies according to the opponent.   There are some opponents who opt to place the ships in the corners, whereas there are some opponents who prefer randomizing the positions. Our main objective is to destroy all the opponent’s ships.

Battleship is known worldwide as a pencil and paper game which dates from World War I. It was released as a plastic board game by Milton Bradley in 1967.We have implemented the same game into a socket programming project. The key features of our game is that, at each stage of the game, a player’s knowledge about its opponent’s board must correspond exactly to what was revealed through faithfully executed game play. This requires concurrent updating of the board after each hit or miss. Furthermore, we have enabled the server to run as a daemon process with logging to syslog. The server is capable of running multiple game sessions with many clients at once.

3. REQUIREMENT ANALYSIS

HARDWARE REQUIREMENTS

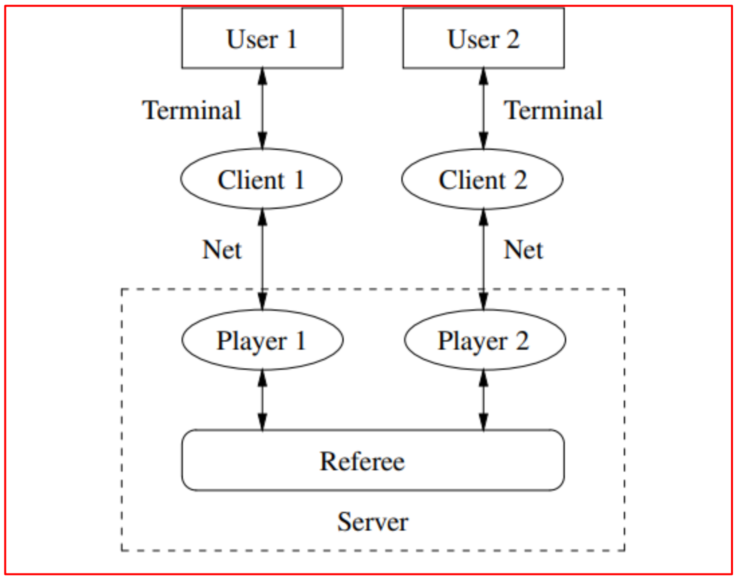
Processor: 2.4 GHz Clock Speed
RAM: Minimum 1 GB
Hard Disk: At Least 100 MB of Free Space

SOFTWARE REQUIREMENTS

Operating System: Windows 7 and above, LINUX
Language: C
Local Platform: Any IDE that runs C. Eg: VsCode
Cloud Platform: Google Colab, AWS (Cloud9)

➔ Compilation is based on the Makefile.

4. ARCHITECTURE



This Illustration describes the modelling of our Program. Our Battleship programs use a client/server architecture in which the interesting computation happens in a multithreaded server. The core of the server is a referee, which takes in network sockets for communicating with the players' client sides, spawns a thread for each player, and then interacts with the players, running the game. The clients simply mediate between the users and the server's player threads, communicating with the users via standard input and output, and with the server via network connections. The Battleship executable is invoked from the Unix/Linux shell, and uses command-line arguments to decide whether to start a server or client. One first starts the server running on a given host/port. Then each of the users starts a client, specifying the host/port to connect to. When game play begins, the first player to have connected will be the first player to shoot. We created specific library files like map.h which we have compiled to do specific tasks like set the grid and ship placements.
It would be easy to write a stand alone client program in which data could flow directly from client to client, not via the server, but that would also interoperate with our server. But our standard client implementation doesn't allow this, despite featuring untrusted code.

- **Defining Program Security**

The server runs as a daemon process with logging to syslog. Therefore, the functions carried out by the server are:

- ❖ A way to start the player, giving it a network socket for communicating with its client side as well as its identity (Player 1 or Player 2).
- ❖ A way of asking the player to choose a complete placing board (CPB), that is where to place each ships. We input this as coordinates of a grid.
- ❖ A way to ask the player what position it wants to shoot next. A way to inform the player of the result of such a shot -illegal repetition, miss, hit of an unspecified ship, sinking of a specified ship. A way to tell the player where its opponent has shot.
- ❖ A way to tell the player it has won or lost the game.

Initially, a CPB is converted to a shooting phase board in which no cells have been shot. And there are functions for:
(a) shooting a cell of a board, returning the shooting result -illegal repetition, miss, hit, sinking of a specified ship- plus the resulting board which is altered according to the result of the shooting;
(b) checking whether all five ships have been sunk on a board. Players will be required to communicate only via their interfaces (including via the network sockets they are passed for communicating with their client sides).
Similarly, the client sides of players will be required to communicate only via their interfaces. They can communicate with their users using I/O and they can communicate with the server side via network socket.
The server is capable of running multiple game sessions with many clients at once. client provides server auto-discovery feature using multicast address (default 224.0.0.200) so you don't need to specify the IP address nor the port when connecting to the server, it works only in the local subnetwork.

We use External Libraries [map.c multicast.c msg.c] that are we wrote for most of the salient features like the ships, defining the map, Functionality of placing the ships on the grid, server auto-discovery and communication between the clients. This simplifies the code as upon compiling and calling each library will reduce the code space.

- **Referee Security**

We say that a server's referee is secure if it communicates only via its interface, doesn't directly use the network sockets for communicating with the players' client sides, and behaves as if it were executing the following model algorithm—as measured from the vantage points of the players:

The referee starts up the players, giving them the network sockets for communicating with their client sides, and telling them their identities player1 and player2.

• The referee obtains CPBs from the players. These CPBs are then converted to shooting phase boards.

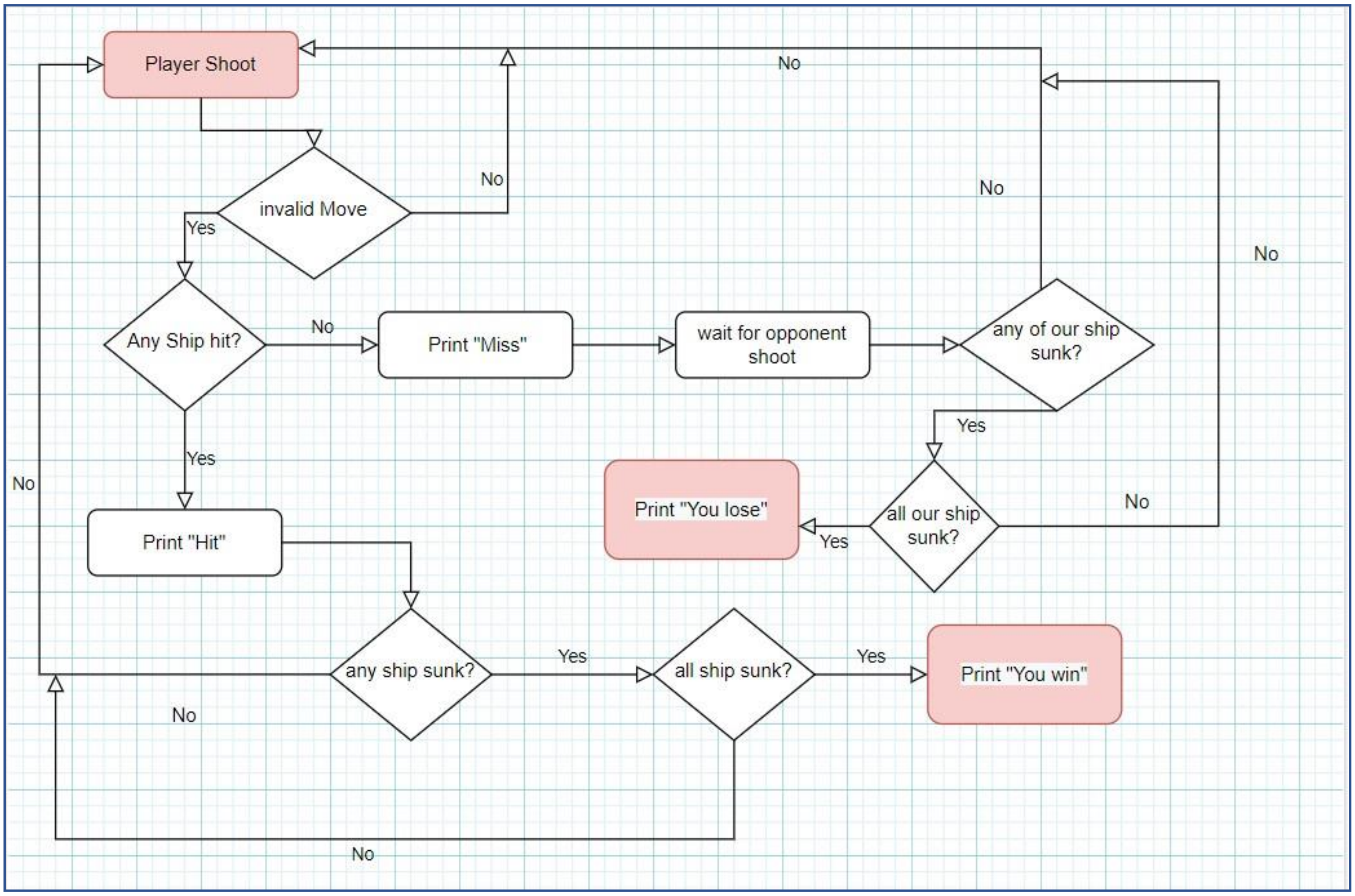
• The referee then enters its main loop, in which it alternates letting the players take shots at each other's boards. Player 1 goes first, then Player 2, etc.

• When all ships of a player's board have been sunk, the player is told it has lost, and its opponent is declared the winner.

• Otherwise, the body of the loop works as follows: The next player to shoot is asked where it wants to shoot on its opponent's board. The shot is carried out using the shooting function, yielding a shooting result and the resulting board. If the shooting result is an illegal repetition, the player is asked to choose a position again, and the opponent's board isn't changed. Otherwise: the player is told the result of its shot; the opponent player is told where on its board the player shot; and the opponent's board is replaced by the board resulting from the shooting.

Because a player can't tell when the referee interacts with its opponent, this definition doesn't impose a total ordering on the referee's interactions with the players. E.g., after calculating the result of a shot, the actions of

- ✓ telling the player, the result of its shot
- ✓ telling the player's opponent where on its board the player shot, could be carried out in either order. On the other hand, the referee can't, e.g., delay action until after it asks the player's opponent where it wants to shoot next, since the player's opponent would notice such a delay.



Because the referee obtains a complete placing board from each player and subsequently manages both players’ boards, part

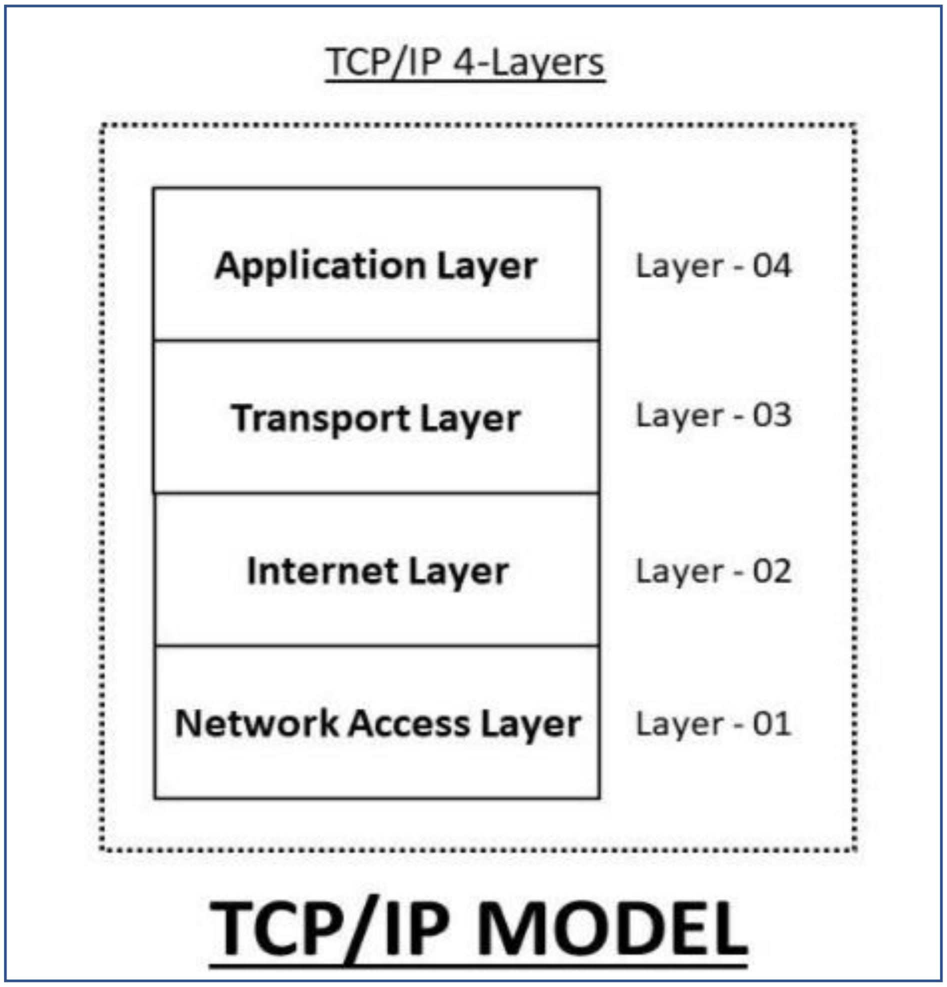
(1) there exists an informal policy is ensured by our definition.

(2) is a consequence of the algorithm followed by the referee. But note our definition’s requirement that the referee keep a player informed of the shots made on its board. This requirement was implicit in the informal policy.

- **TCP/IP Protocol Architecture**

It is a four-layered protocol stack. It helps in the interconnection of network devices over the internet. Each layer contains certain protocols that help in the functioning of the layer. The four layers of TCP/IP protocol are Application Layer, Transport Layer, Networking/Internet Layer and the Data Link/physical layer.

A socket programming interface provides the routines required for interprocess communication between applications, either on the local system or spread in a distributed, TCP/IP based network environment. Once a peer-to-peer connection is established, a socket descriptor is used to uniquely identify the connection. The socket descriptor itself is a task specific numerical value.One end of a peer-to-peer connection of a TCP/IP based distributed network application described by a socket is uniquely defined by
Internet address-for example 127.0.0.1 (in an IPv4 network) or FF01::101 (in an IPv6 network).
Port-A numerical value, identifying an application. We distinguish between "well known" ports, for example port 23 for Telnet user defined ports.



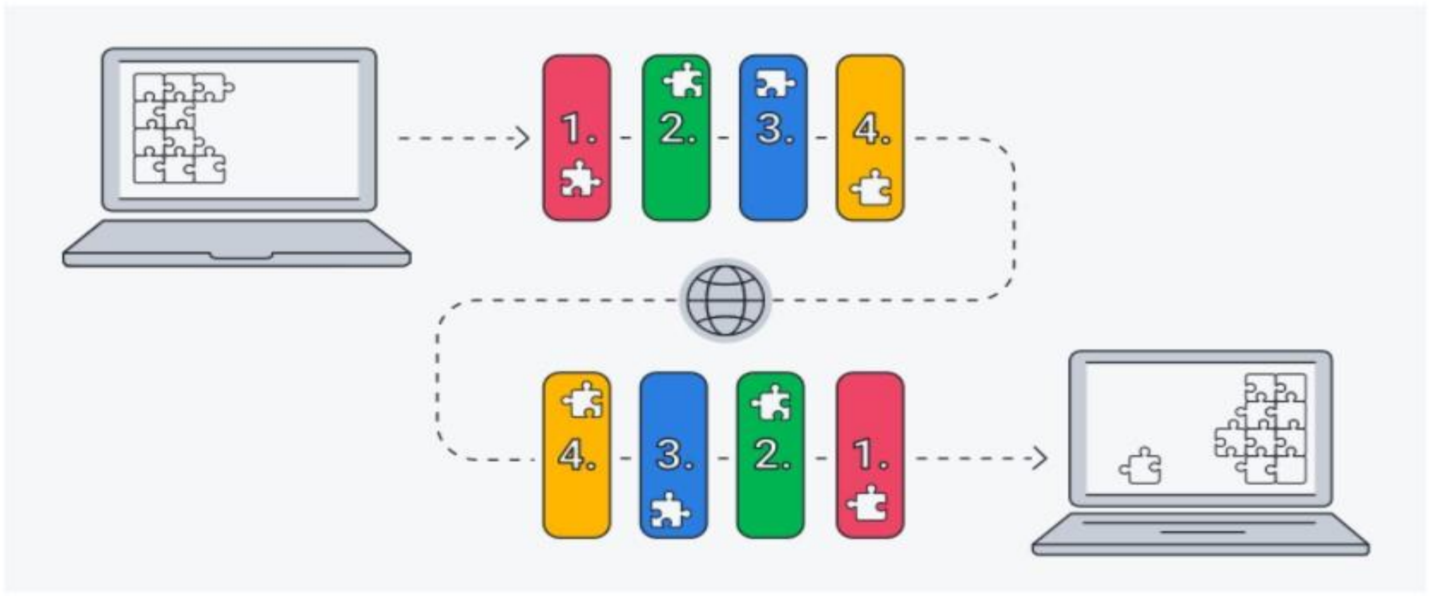
- **How it works**

TCP/IP is a two-layered program: To ensure that each communication reaches its intended destination intact, the TCP/IP model breaks down data into packets and then reassembles the packets into the complete message on the other end. Sending the data in small packets makes it easier to maintain accuracy versus sending all the data at once.

After a single message is split into packets, these packets may travel along different routes if one route is congested. It's like sending a few different birthday cards to the same household by mail. The cards begin their journey at your home, but you might drop each card into a different mailbox, and each card may take a different path to the recipient's address.

the higher layer (TCP) disassembles message content into small "data packets" that are then transmitted over the Internet to be re-assembled by the receiving computer's TCP back into the message's original form. The lower layer (IP) plays the role of "address manager" and gets each data packet to the correct destination. IP addresses are checked by every computer in a network to ensure messages are forwarded as needed.

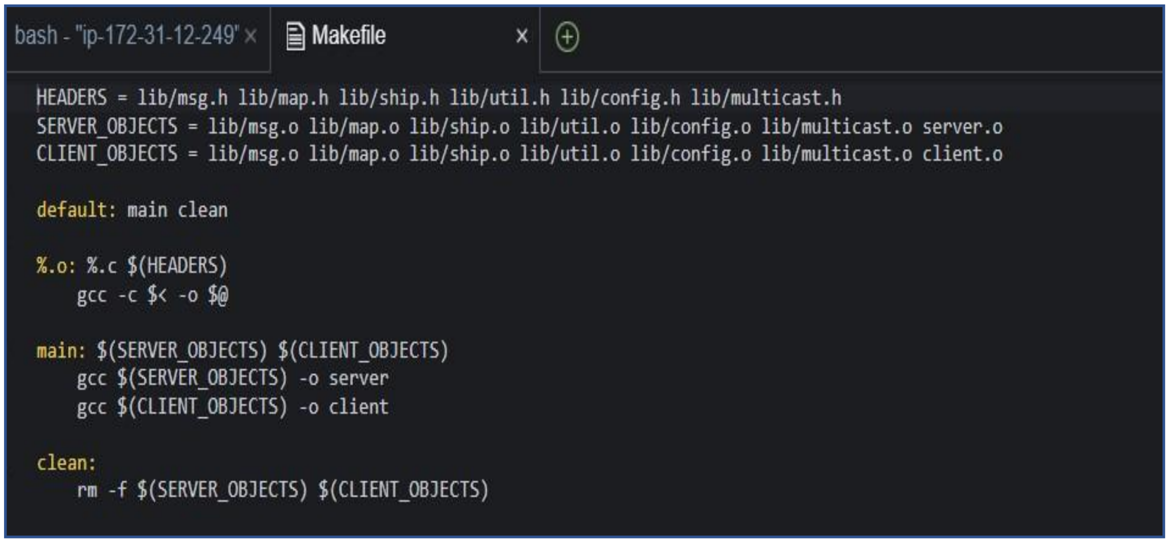
TCP/IP also relies on point-to-point communication, meaning that communications move from one host computer to another within a pre-defined network boundary. TCP/IP is said to be stateless because each request is new and unrelated to all previous requests, making network pathways free to be continuously used by all.



A diagram of how the TCP/IP model divides data into packets and sends it through 4 different layers.

5. IMPLEMENTATION

To implement this, we require Amazon AWS. We should first call the makefile. Makefile automatically traverses and lists all the code files in sub-directories, not only for executable binaries, but also for static & dynamic libraries.



```
HEADERS = lib/msg.h lib/map.h lib/ship.h lib/util.h lib/config.h lib/multicast.h
SERVER_OBJECTS = lib/msg.o lib/map.o lib/ship.o lib/util.o lib/config.o lib/multicast.o server.o
CLIENT_OBJECTS = lib/msg.o lib/map.o lib/ship.o lib/util.o lib/config.o lib/multicast.o client.o

default: main clean

%.o: %.c $(HEADERS)
    gcc -c $< -o $@

main: $(SERVER_OBJECTS) $(CLIENT_OBJECTS)
    gcc $(SERVER_OBJECTS) -o server
    gcc $(CLIENT_OBJECTS) -o client

clean:
    rm -f $(SERVER_OBJECTS) $(CLIENT_OBJECTS)
```

In this makefile, we have incorporated the build of all the external libraries ,like map.h, ship.h, which are all stored in a specific folder library. The makefile also compiles both the server and client code. We can execute the makefile by typing make.

This will execute ./server in the background, then call ./ client multiple times, with a variety of command line arguments. For this assessment IPC model was chosen and implemented with unix sockets, and multithreaded my main node (./server), so that it is able to accept multiple client requests simultaneously. We ensure atomicity of operations and proper synchronization using pthread_mutex_locks.

./server is the master node executable which is capable of accepting multiple connections from clients simultaneously. ./ client takes command line arguments that specify an operation specified by a player for the server to process. The output of make test consists of the commands which execute the executables that were compiled by make all to demonstrate the functionality of this project: then, when the majority of the basic functionality the executables are capable of are gone through, the log_file that is written to every time a client operation takes place on the server node is then displayed with the cat command on the terminal, so the tester can see the result of the commands.

After the basic operation of the executables are demonstrated, a SIGINT is sent to the master node to stop it. The master node then executes its interrupt routine for the SIGINT signal and the server/game is done.

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <syslog.h>
#include <signal.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>

#include "lib/msg.h"
#include "lib/map.h"
#include "lib/ship.h"
#include "lib/util.h"
#include "lib/config.h"
#include "lib/multicast.h"

int num_of_games = 0;
void sig_chld(int signo)
{    pid_t   pid;
    int     stat;
    while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
        syslog (LOG_INFO,"Child %d terminated\n", pid);
    return;
}
void error(const char *msg)
{    syslog (LOG_ERR, msg);
    exit(0);  }
int setup_listener(int portno)
{    int sockfd;
    struct sockaddr_in serv_addr;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening listener socket.");
    int enable = 1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)
        error("ERROR setsockopt() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR binding listener socket.");
    syslog (LOG_INFO,"Listener set.");
    return sockfd;
}
```

```
void get_clients(int lis_sockfd, int * cli_sockfd)
{    socklen_t clilen;
    struct sockaddr_in serv_addr, cli_addr;
    id_message id_msg;
    hold_message hold_msg;
        syslog (LOG_INFO,"Waiting for clients...");
    int num_conn = 0;
    while(num_conn < 2)
            listen(lis_sockfd, 2);
        memset(&cli_addr, 0, sizeof(cli_addr));
        clilen = sizeof(cli_addr);
        cli_sockfd[num_conn] = accept(lis_sockfd, (struct sockaddr *) &cli_addr, &clilen);
            if (cli_sockfd[num_conn] < 0)
            error("ERROR accepting a connection from a client.");
        syslog (LOG_INFO,"Accepted connection from client %d", num_conn);
        if(send_id_message(cli_sockfd[num_conn], num_conn) < 0)
            error("ERROR sending ID to client");
        syslog (LOG_INFO,"Sent client %d it's ID.", num_conn);
        if (num_conn == 0) {
            if(send_hold_message(cli_sockfd[0]) < 0)
                error("ERROR sending HOLD to client");
                        syslog (LOG_INFO,"Told client 0 to hold.");
        }         num_conn++;      }  }
int get_clients_coords(int *cli_sockfd, Map* map_p0, Map* map_p1){
    void* message = NULL;
    int msg_type;
    int ships_left_p0 = 0;
    int ships_left_p1 = 0;
    int hold_sent = 0;
    int maxfd, n;
    fd_set rset;
    FD_ZERO(&rset);
    if(cli_sockfd[0]>cli_sockfd[1])
        maxfd = cli_sockfd[0];
    else
        maxfd = cli_sockfd[1];
    syslog (LOG_INFO,"Waiting for players' ship coordinates");
    do
    {  syslog (LOG_INFO,"Waiting for INSERT messages from clients...");
        FD_SET(cli_sockfd[0], &rset);
        FD_SET(cli_sockfd[1], &rset);
        if ( ( n = select(maxfd+1, &rset, NULL, NULL, NULL) ) < 0){
            return -4;
        }
        syslog (LOG_INFO,"%d sockets ready for read.\n", n);
        if (FD_ISSET(cli_sockfd[0], &rset)) {
            syslog (LOG_INFO,"Socket 0 is ready");
            if( (msg_type = receive_message(cli_sockfd[0], &message) ) < 0 )
                error("ERROR receiving message form client");
        }
        if (FD_ISSET(cli_sockfd[1], &rset)) {
            syslog (LOG_INFO,"Socket 1 is ready");
            if( (msg_type = receive_message(cli_sockfd[1], &message) ) < 0 )
                error("ERROR receiving message form client");
        }
        if(msg_type == INSERT_MSG_TYPE){
            syslog (LOG_INFO,"recived INSERT message");
            if (insert_ship(
                ((insert_message*)message)->id ? map_p1 : map_p0,
                ((insert_message*)message)->ship,
                ((insert_message*)message)->x,
                ((insert_message*)message)->y,
                ((insert_message*)message)->orientation
                )== -1)
                error("ERROR inserting ship based on recived client data");
            ships_left_p0 = check_used_ships(map_p0);
            ships_left_p1 = check_used_ships(map_p1);
            syslog (LOG_INFO,"Ships left to assign: Player0=%d, Player1=%d\n", ships_left_p0, ships_left_p1);
            if(ships_left_p0 == 0){
                syslog (LOG_INFO,"Recived all player0's coordinates");
                if(!hold_sent){
                    if(send_hold_message(cli_sockfd[0]) < 0)
                        error("ERROR sending HOLD to client");
                    hold_sent = 1;
                    syslog (LOG_INFO,"sent HOLD to player0");
                }           }
            if(ships_left_p1 == 0){
                syslog (LOG_INFO,"Recived all player1's coordinates");
                if(!hold_sent){
                    if(send_hold_message(cli_sockfd[1]) < 0)
                        error("ERROR sending HOLD to client");
                    hold_sent = 1;
                    syslog (LOG_INFO,"sent HOLD to player1");
                }           }
                    }else
            error("ERROR wrong message type, expected INSERT message");
    }while(ships_left_p0 > 0 || ships_left_p1 > 0);
    syslog (LOG_INFO,"Both players' ship coordinates recived. Starting the game...");
    return 1;
}
void run_game(int *cli_sockfd )
{   syslog (LOG_INFO,"GAME ON!");
    Map *map_p0 = init_map_matrix(MAP_WIDTH, MAP_HEIGH);
    Map *map_p1 = init_map_matrix(MAP_WIDTH, MAP_HEIGH);
    if(send_start_message(cli_sockfd[0]) < 0)
        error("ERROR sending START to client1");
    if(send_start_message(cli_sockfd[1]) < 0)
        error("ERROR sending START to client2");
    syslog (LOG_INFO,"Sent start message.");
    get_clients_coords( cli_sockfd, map_p0, map_p1);
    srand(time(NULL));
    int starting_id = rand() % 2;
    if(send_begin_message(cli_sockfd[0], starting_id) < 0)
        error("ERROR sending BEGIN to client1");
    if(send_begin_message(cli_sockfd[1], starting_id) < 0)
        error("ERROR sending BEGIN to client2");
    show_maps(map_p0, map_p1);
    int game_over = 0;
    int player_turn = starting_id;
    int msg_type, code;
    void* message = NULL;
    while(!game_over) {
        syslog (LOG_INFO,"Waiting for Player%d's move\n", player_turn);
        if( (msg_type = receive_message(cli_sockfd[player_turn], &message) ) < 0 )
                error("ERROR receiving message form client");
        if(msg_type == ATTACK_MSG_TYPE){
            syslog (LOG_INFO,"Recived ATTACK message");
            Map *map = ((attack_message*)message)->id ? map_p0 : map_p1;
            if ( ( code = attack_ship(map, ((attack_message*)message)->x, ((attack_message*)message)->y) ) ){
                if (check_map(map) == 0)
                {                   syslog (LOG_INFO,"GAME OVER!");
```

```
                    if(send_status_message(cli_sockfd[0], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, GAMEOVER, 0) < 0)
                        error("ERROR sending STATUS to client");
                    if(send_status_message(cli_sockfd[1], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, GAMEOVER, 0) < 0)
                        error("ERROR sending STATUS to client");
                    syslog (LOG_INFO,"Sent STATUS message");
                    break;
                }
                else
                {
                    if(code == 1){
                        syslog (LOG_INFO,"HIT");
                        if(send_status_message(cli_sockfd[0], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, HIT, 0) < 0)
                            error("ERROR sending STATUS to client");
                        if(send_status_message(cli_sockfd[1], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, HIT, 0) < 0)
                            error("ERROR sending STATUS to client");
                        syslog (LOG_INFO,"Sent STATUS message");
                    }else if(code == 2){
                        syslog (LOG_INFO,"SUNK");
                        int ship_type = getType(map, ((attack_message*)message)->x, ((attack_message*)message)->y );
                        if(send_status_message(cli_sockfd[0], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, SUNK, ship_type) < 0)
                            error("ERROR sending STATUS to client");
                        if(send_status_message(cli_sockfd[1], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, SUNK, ship_type) < 0)
                            error("ERROR sending STATUS to client");
                        syslog (LOG_INFO,"Sent STATUS message");
                    }
                }
            }else{
                syslog (LOG_INFO,"MISS");
                if(send_status_message(cli_sockfd[0], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, MISS, 0) < 0)
                    error("ERROR sending STATUS to client");
                if(send_status_message(cli_sockfd[1], ((attack_message*)message)->id, ((attack_message*)message)->x,
((attack_message*)message)->y, MISS, 0) < 0)
                    error("ERROR sending STATUS to client");
                syslog (LOG_INFO,"Sent STATUS message");
                player_turn = !player_turn; //change player turn
            }
            show_maps(map_p0, map_p1);
        }
        else
            error("ERROR wrong message type, expected ATTACK message");
    }
    syslog (LOG_INFO,"QUITTING...");
    close(cli_sockfd[0]);
    close(cli_sockfd[1]);
        free(cli_sockfd);
}
void service_discovery()
{
    int sendfd, recvfd;
    const int on = 1;
    socklen_t salen;
    struct sockaddr *sasend, *sarecv;
    struct sockaddr_in6 *ipv6addr;
    struct sockaddr_in *ipv4addr;
    char    *addr_str;
    sendfd = snd_udp_socket(SERVICE_MULTICAST_ADDR, SERVICE_PORT, &sasend, &salen);
    if ( (recvfd = socket(sasend->sa_family, SOCK_DGRAM, 0)) < 0){
        error("ERROR: socket error");
    }     if (setsockopt(recvfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0){
        error("ERROR: setsockopt error");
    }     sarecv = malloc(salen);
    memcpy(sarecv, sasend, salen);
     if(sarecv->sa_family == AF_INET6){
      ipv6addr = (struct sockaddr_in6 *) sarecv;
      ipv6addr->sin6_addr =  in6addr_any;
    }
   if(sarecv->sa_family == AF_INET){
      ipv4addr = (struct sockaddr_in *) sarecv;
      ipv4addr->sin_addr.s_addr =  htonl(INADDR_ANY);
    }
     if( bind(recvfd, sarecv, salen) < 0 ){
        error("ERROR: bind error");
    }
     if( mcast_join(recvfd, sasend, salen, NULL, 0) < 0 ){
        error("ERROR: mcast_join() error");
    }
    mcast_set_loop(sendfd, 1);
    while(1)
        recv_multicast(recvfd, salen);
}
#define MAXFD    64
int daemon_init(const char *pname, int facility, uid_t uid, int socket){
    int      i, p;
    pid_t    pid;
    if ( (pid = fork()) < 0)
        return (-1);
    else if (pid)
        exit(0
    if (setsid() < 0
        return (-1);
    signal(SIGHUP, SIG_IGN);
    if ( (pid = fork()) < 0)
        return (-1);
    else if (pid)
        exit(0
    chdir("/tmp
    for (i = 0; i < MAXFD; i++){
        if(socket != i )
            close(i);
    }
    p= open("/dev/null", O_RDONLY);
    open("/dev/null", O_RDWR);
    open("/dev/null", O_RDWR);
    openlog(pname, LOG_PID, facility);
            syslog(LOG_ERR," STDIN =    %i\n", p);
    setuid(uid);
        return (0);
int main(int argc, char *argv[])
{            signal(SIGCHLD, sig_chld);
        daemon_init(argv[0], LOG_USER, 1000, MAXFD);
    syslog (LOG_NOTICE, "Program started by User %d", getuid ());
    if ( fork() == 0) {
        service_discovery();
        exit(0);
    }
    syslog (LOG_INFO,"Setting up listener...");
    int lis_sockfd = setup_listener(SERVICE_PORT);
    while (1) {
        if( (num_of_games++) < 10) {
            int *cli_sockfd = (int*)malloc(2*sizeof(int));
            get_clients(lis_sockfd, cli_sockfd);
            syslog (LOG_INFO,"Starting new game thread... ");
            if ( fork() == 0)
                close(lis_sockfd);
                run_game(cli_sockfd);
                exit(0);
            }
            close(cli_sockfd[0]);
            close(cli_sockfd[1]);
            syslog (LOG_INFO,"New game thread started.");
        }   }
    close(lis_sockfd);
    syslog (LOG_INFO,"quitting...");
}
```

Client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include "lib/msg.h"
#include "lib/map.h"
#include "lib/ship.h"
#include "lib/util.h"
#include "lib/config.h"
#include "lib/multicast.h"
#define CLEAR
void error(const char *msg)
{  #ifdef DEBUG
    perror(msg);
    #else
    printf("Either the server shut down or the other player disconnected.\nGame over.\n");
    #endif
    exit(0);
}
int setup_udp_listener(int portno)
{    int sockfd;
    struct sockaddr_in serv_addr;
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0)
        error("ERROR opening listener socket.");
    int enable = 1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)
        error("ERROR setsockopt() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR binding listener socket.");
    #ifdef DEBUG
    printf("[DEBUG] UDP Listener set.\n");
    #endif }
int connect_to_server(char * hostname, int portno)
{    struct sockaddr_in serv_addr;
    int err;
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
                if (sockfd < 0)
        error("ERROR opening socket for server.");
memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(portno);
    if ( (err=inet_pton(AF_INET, hostname, &serv_addr.sin_addr)) == -1){
        error("ERROR: inet_pton error");
    }else if(err == 0){
        error("ERROR: Invalid address family");
    }
   if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting to server");
    #ifdef DEBUG
    printf("[DEBUG] Connected to server.\n");
    #endif
       return sockfd;
}
int attack(int sockfd, int id){
    int x, y;
    printf("Your turn to attack\n");
    printf("x: ");
    scanf("%i", &x);
    while(x > MAP_WIDTH || x < 0){
        printf("Invalid choice.");
        scanf("%i", &x);
    }
    printf("y: ");
    scanf("%i", &y);
    while(y > MAP_HEIGH || y < 0){
        printf("Invalid choice.");
        scanf("%i", &x);
    }
    if(send_attack_message(sockfd, id, x, y) < 0)
        error("ERROR sending ATTACK message");
    #ifdef DEBUG
    printf("[DEBUG] sent ATTACK message\n");
    #endif
}
int main(int argc, char *argv[])
{    #ifdef RANDOMIZE
    srand(time(NULL));
    #endif
       int sendfd, recvfd, n;
    socklen_t salen, len;
    struct sockaddr *sasend, *safrom;
    struct sockaddr_in6 *cliaddr;
    struct sockaddr_in *cliaddrv4;
    char   addr_str[INET6_ADDRSTRLEN+1];
    safrom = malloc(salen);
```

```
recvfd = setup_udp_listener(2222);
sendfd = snd_udp_socket(SERVICE_MULTICAST_ADDR, SERVICE_PORT, &sasend, &salen);
send_multicast(sendfd, sasend, salen);
#ifdef DEBUG
printf("[DEBUG] sending multicast request\n");
#endif
len = sizeof(safrom);
if( (n = recvfrom(recvfd, NULL, 0, 0, safrom, &len)) < 0 )
      perror("recvfrom() error");
        if( safrom->sa_family == AF_INET6 ){
      cliaddr = (struct sockaddr_in6*) safrom;
      inet_ntop(AF_INET6, (struct sockaddr  *) &cliaddr->sin6_addr,  addr_str, sizeof(addr_str));
}
else{
      cliaddrv4 = (struct sockaddr_in*) safrom;
      inet_ntop(AF_INET, (struct sockaddr  *) &cliaddrv4->sin_addr,  addr_str, sizeof(addr_str));
}
#ifdef DEBUG
printf("[DEBUG] Recived reply from: %s\n", addr_str);
#endif
close(recvfd);
close(sendfd);
int sockfd = connect_to_server(addr_str, SERVICE_PORT);
printf("Connected.\n");
int id;
void* message = NULL;
int msg_type;
if( (msg_type = receive_message(sockfd, &message) ) < 0 )
    error("ERROR receiving ID form server");
else
    if(msg_type == ID_MSG_TYPE){
        id = ((id_message*)message)->id;
        #ifdef DEBUG
        printf("[DEBUG] received ID message\n");
        #endif
    }else
        error("ERROR wrong message type, expected id_message");
    #ifdef DEBUG
printf("[DEBUG] Client ID: %d\n", id);
#endif
do {
    if( (msg_type = receive_message(sockfd, &message) ) < 0 ){
        error("ERROR receiving message form server");
    }
    else
        if(msg_type == HOLD_MSG_TYPE){
            #ifdef DEBUG
            printf("[DEBUG] received HOLD message\n");
            #endif
            printf("Waiting for a second player...\n");
            }
} while ( msg_type != START_MSG_TYPE );
#ifdef DEBUG
printf("[DEBUG] received START message\n");
#endif
printf("Game on!\n");
sleep(1);
#ifdef CLEAR
system("clear");
#endif
Map *my_map = init_map_matrix(MAP_WIDTH, MAP_HEIGH);
Map *opponent_map = init_map_matrix(MAP_WIDTH, MAP_HEIGH);
int i, ship, x, y, orientation; // variables used for ship insertion;
while((i = check_used_ships(my_map)) > 0)
{  #ifdef CLEAR
    system("clear");
    #endif
    show_map(my_map);
    show_ships(my_map);
    printf("You still have %i ship(s) to organize\n", i);
    printf("Choose one to put in the map: ");
    #ifdef RANDOMIZE
    ship = rand() % 4;
    #else
    scanf("%i", &ship);
    #endif
    while( (ship > 3 || ship < 0) || my_map->ships[ship] <= 0)
    {   #ifdef CLEAR
        system("clear");
        show_map(my_map);
        show_ships(my_map);
        printf("You still have %i ship(s) to organize\n", i);
        #endif
        printf("Invalid choice\n");
        printf("Please, choose another one: ");
        #ifdef RANDOMIZE
        ship = rand() % 4;
        #else
        scanf("%i", &ship);
        #endif
    }
    printf("Orientation (1-vert/2-hori): ");
    #ifdef RANDOMIZE
    orientation = 1 + rand() % 2;
    #else
    scanf("%i", &orientation);
    #endif
    while(orientation != 1 && orientation != 2)
    {   #ifdef CLEAR
        system("clear");
        show_map(my_map);
        show_ships(my_map);
        printf("Orientation (1-vert/2-hori): ");
        #endif
        printf("Invalid choice\n");
        printf("Please, choose another option: ");
        #ifdef RANDOMIZE
        orientation = 1 + rand() % 2;
        #else
        scanf("%i", &orientation);
        #endif
    }
    printf("Ship head position\n");
    printf("x: ");
    #ifdef RANDOMIZE
    x = rand() % 10;
    #else
    scanf("%i", &x);
    #endif
    while(x > my_map->width || x < 0)
    {  #ifdef CLEAR
        system("clear");
        show_map(my_map);
        show_ships(my_map);
        printf("Ship head position\n");
        printf("x: ");
        #endif
        printf("Invalid choice\n");
        printf("Please, choose another x: ");
        #ifdef RANDOMIZE
        x = rand() % 10;
        #else
        scanf("%i", &x);
        #endif
    }
    printf("y: ");
    #ifdef RANDOMIZE
    y = rand() % 10;
    #else
    scanf("%i", &y);
    #endif
    while(y > my_map->height || y < 0)
    {  #ifdef CLEAR
        system("clear");
        show_map(my_map);
        show_ships(my_map);
        printf("Ship head position\n");
        printf("y: ");
        #endif
        printf("Invalid choice\n");
        printf("Please, choose another y: ");
        #ifdef RANDOMIZE
        y = rand() % 10;
        #else
        scanf("%i", &y);
        #endif
    }
    if (insert_ship(my_map, ship, x, y, orientation) == -1)
        printf("\nOut of limit!\nChoose again...\n\n");
    else
        if(send_insert_message(sockfd, id, ship, x, y, orientation) < 0)
            error("ERROR sending INSERT to server");
        else{
            #ifdef DEBUG
            printf("[DEBUG] sent INSERT message\n");
            #endif
        }    }
#ifdef CLEAR
system("clear");
#endif
printf("\nShips ready!\n");
show_maps(my_map, opponent_map);
show_ships_left(opponent_map);
do {
    if( (msg_type = receive_message(sockfd, &message) ) < 0 ){
        error("ERROR receiving message form server");
    }
    else  if(msg_type == HOLD_MSG_TYPE){
            #ifdef DEBUG
            printf("[DEBUG] received HOLD message\n");
            #endif
            printf("Waiting for a second player...\n");                    }
} while ( msg_type != BEGIN_MSG_TYPE );
```

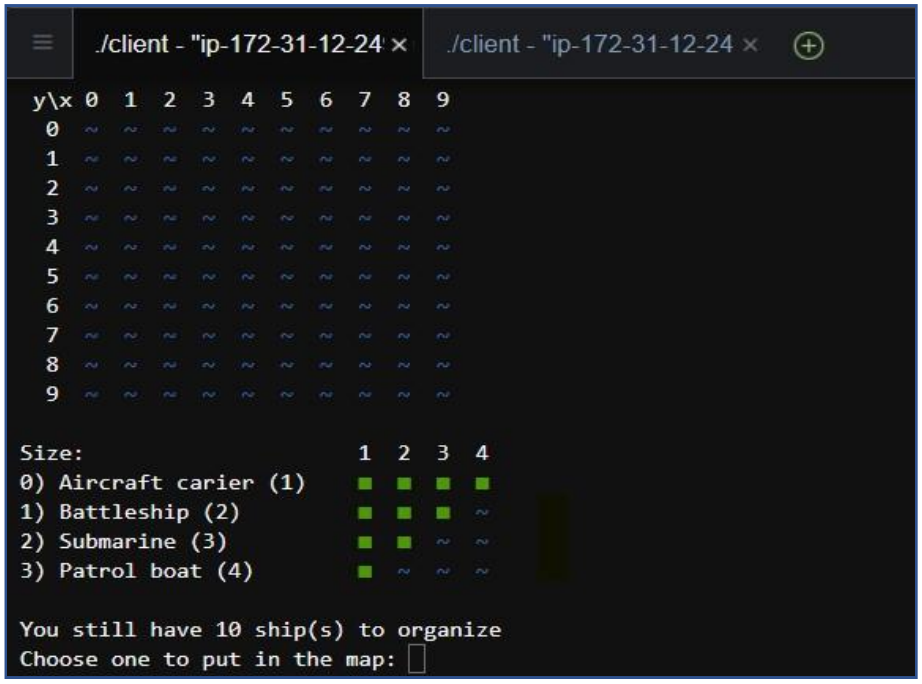
```
#ifdef DEBUG
    printf("[DEBUG] recived BEGIN message\n");
#endif
#ifdef CLEAR
system("clear");
show_maps(my_map, opponent_map);
show_ships_left(opponent_map);
#endif
if (id == ((begin_message*)message)->id)
    attack(sockfd, id);
else
    printf("Oponent's move. Waiting...\n");
  int ship_type;
while(1) {
    if( (msg_type = receive_message(sockfd, &message) ) < 0 ){
        error("ERROR receiving message form server");
    }
    else
        if(msg_type == STATUS_MSG_TYPE){
            #ifdef DEBUG
            printf("[DEBUG] received STATUS message\n");
            #endif
            if( id == ((status_message*)message)->id ){
                switch (((status_message*)message)->response)
                {       case MISS:
                        opponent_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = MISSED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("Missed. Oponent's move...\n");
                        break;
                    case HIT:
                        opponent_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("You hit.\n");
                        attack(sockfd, id);
                        break;
                    case SUNK:
                        opponent_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        ship_type = ((status_message*)message)->options;
                        opponent_map->ships[ship_type]--;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("Hit and sunk!\n");
                        attack(sockfd, id);
                        break;
                    case GAMEOVER:
                        opponent_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        ship_type = ((status_message*)message)->options;
                        opponent_map->ships[ship_type]--;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        PRINT_BLUE("\nYOU WON!!!\n\n");
                        close(sockfd);
                        return 0;
                    default:
                        break;
                }
            }else{
                switch (((status_message*)message)->response)
                {
                    case MISS:
                        my_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = MISSED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("Oponent missed.\n");
                        attack(sockfd, id);
                        break;
                    case HIT:
                        my_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("Oponent hit. Waiting for his next move...\n");
                        break;
                    case SUNK:
                        my_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        printf("Oponent has hit and sunk your ship!\n");
                        break;
                    case GAMEOVER:
                        my_map->map[((status_message*)message)->y][((status_message*)message)->x][0] = DESTROYED;
                        #ifdef CLEAR
                        system("clear");
                        #endif
                        show_maps(my_map, opponent_map);
                        show_ships_left(opponent_map);
                        PRINT_RED("\nYOU LOST!!!\n\n");
                        close(sockfd);
                        return 0;
                    default:
                        break;
                }
            }
        }else
            error("ERROR wrong message type, expected ATTACK message");
      }
      printf("Game over.\n");
      close(sockfd);
      return 0;
    }
```

6. EXPERIMENT RESULTS & ANALYSIS

6.1. RESULTS

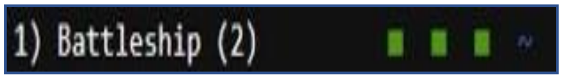
Player 1 perspective

1. **Place Ships**



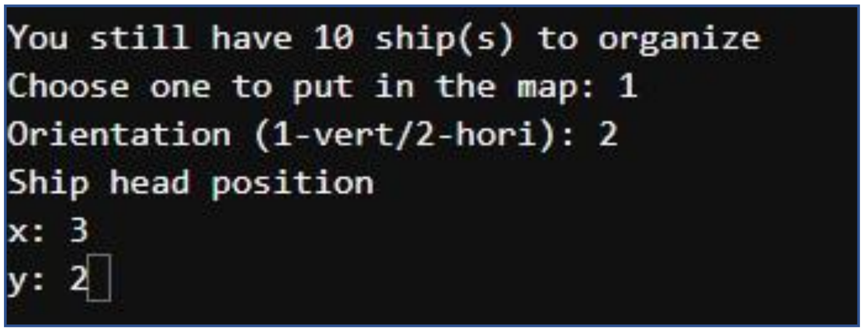
```
./client - "ip-172-31-12-24 ×   ./client - "ip-172-31-12-24 ×

 y\x 0  1  2  3  4  5  6  7  8  9
  0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                         1  2  3  4
0) Aircraft carier (1)        ■  ■  ■  ■
1) Battleship (2)             ■  ■  ■  ~
2) Submarine (3)              ■  ■  ~  ~
3) Patrol boat (4)            ■  ~  ~  ~

You still have 10 ship(s) to organize
Choose one to put in the map:
```

- Click 0 for aircraft carrier,1 for battleship, 2 for submarine 3 for patrol boat.



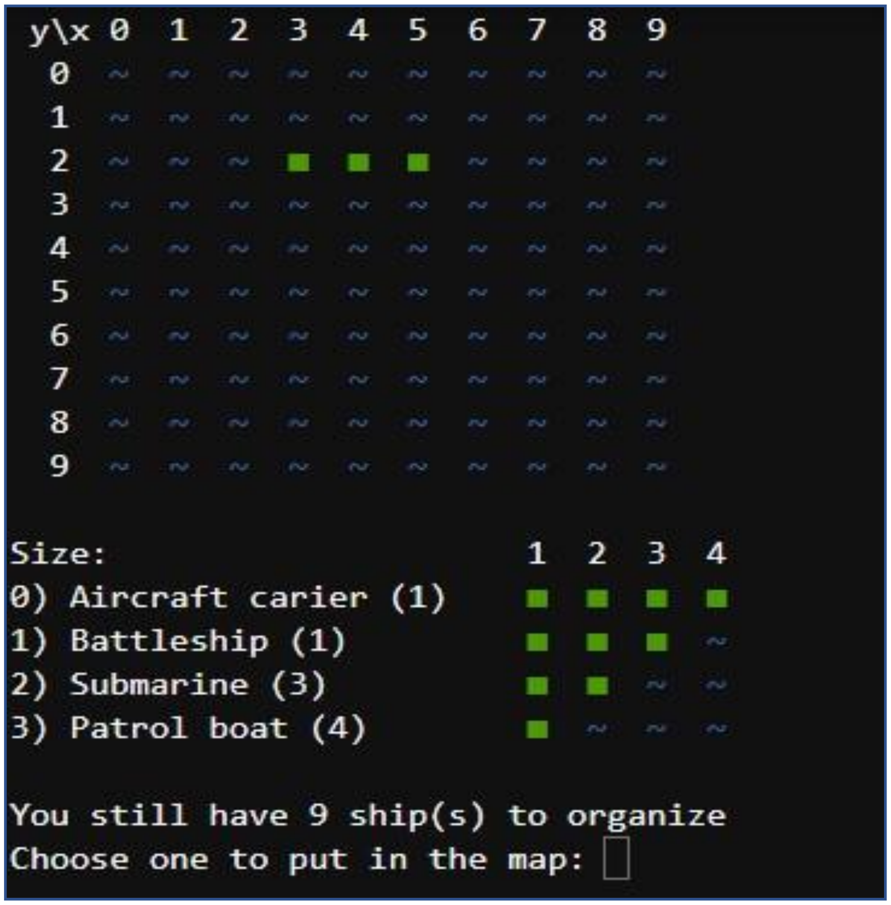
```
1) Battleship (2)        ■  ■  ■  ~
```

- Here **1)** refers to the number to be inputted to select which type of ship.
- **Battleship** is the name of the ship.
- **(2)** shows number of battleship left to be placed.
- represents the size of the battleship.



```
You still have 10 ship(s) to organize
Choose one to put in the map: 1
Orientation (1-vert/2-hori): 2
Ship head position
x: 3
y: 2
```

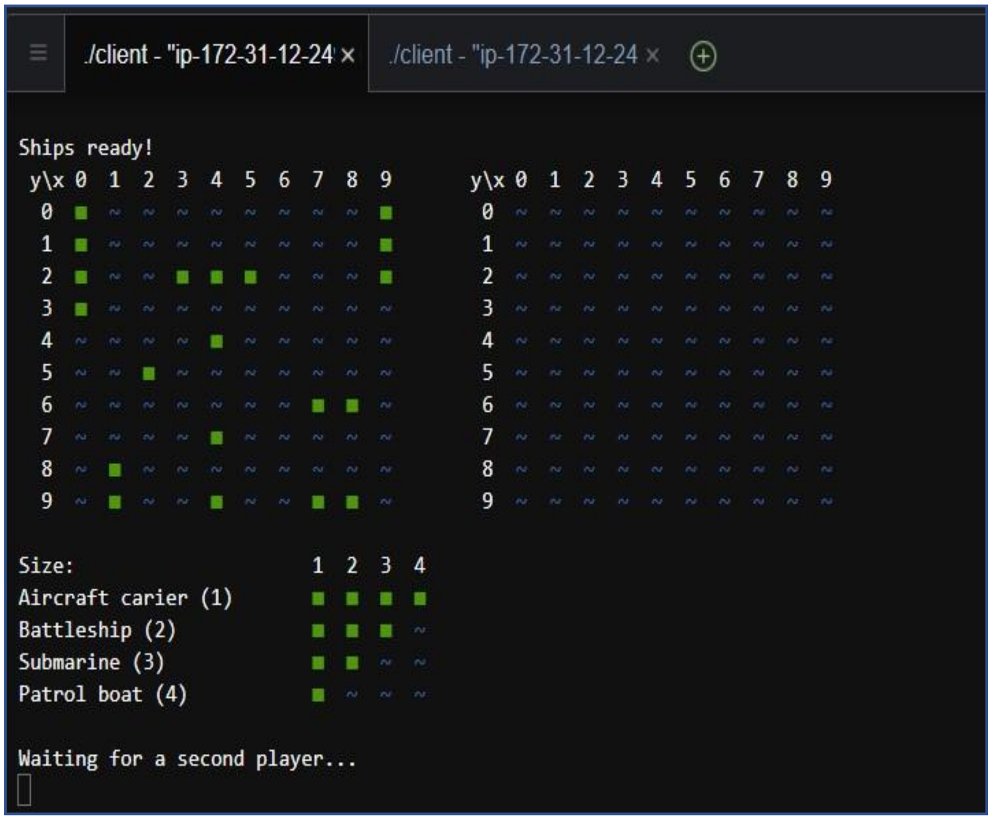
- We have chosen Battleship (1). Orientation refers to the placement of the ship (should be kept horizontally or vertically)
- Here we have chosen horizontally.
- Input ship position coordinate.



```
 y\x 0  1  2  3  4  5  6  7  8  9
  0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  2  ~  ~  ~  ■  ■  ■  ~  ~  ~  ~
  3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                     1  2  3  4
0) Aircraft carier (1)    ■  ■  ■  ■
1) Battleship (1)         ■  ■  ■  ~
2) Submarine (3)          ■  ■  ~  ~
3) Patrol boat (4)        ■  ~  ~  ~

You still have 9 ship(s) to organize
Choose one to put in the map:
```

- First ship Placed, nine more to be placed.
- Keep in mind, the size of each type of ship varies. For example, Aircraft carrier is of size 4, whereas Submarine is of size 2.

2. Ships are ready waiting for another player

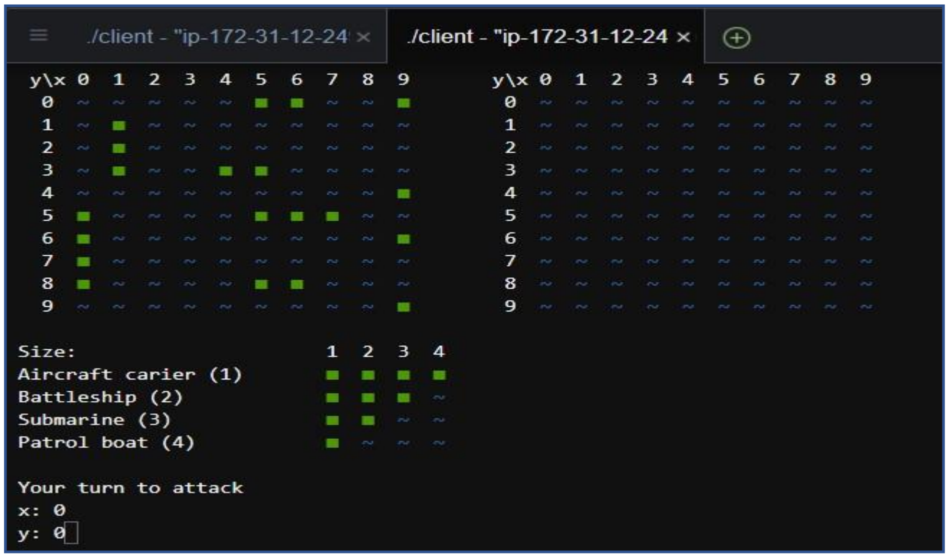


./client - "ip-172-31-12-24 ×   ./client - "ip-172-31-12-24 ×

```
Ships ready!
 y\x 0  1  2  3  4  5  6  7  8  9       y\x 0  1  2  3  4  5  6  7  8  9
  0  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■        0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  1  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■        1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  2  ■  ~  ~  ■  ■  ■  ~  ~  ~  ■        2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  3  ■  ~  ~  ~  ~  ~  ~  ~  ~  ~        3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  4  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~        4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  5  ~  ~  ■  ~  ~  ~  ~  ~  ~  ~        5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  6  ~  ~  ~  ~  ~  ~  ~  ■  ■  ~        6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  7  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~        7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  9  ~  ■  ~  ~  ■  ~  ~  ■  ■  ~        9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                  1  2  3  4
Aircraft carier (1)    ■  ■  ■  ■
Battleship (2)         ■  ■  ■  ~
Submarine (3)          ■  ■  ~  ~
Patrol boat (4)        ■  ~  ~  ~

Waiting for a second player...
```

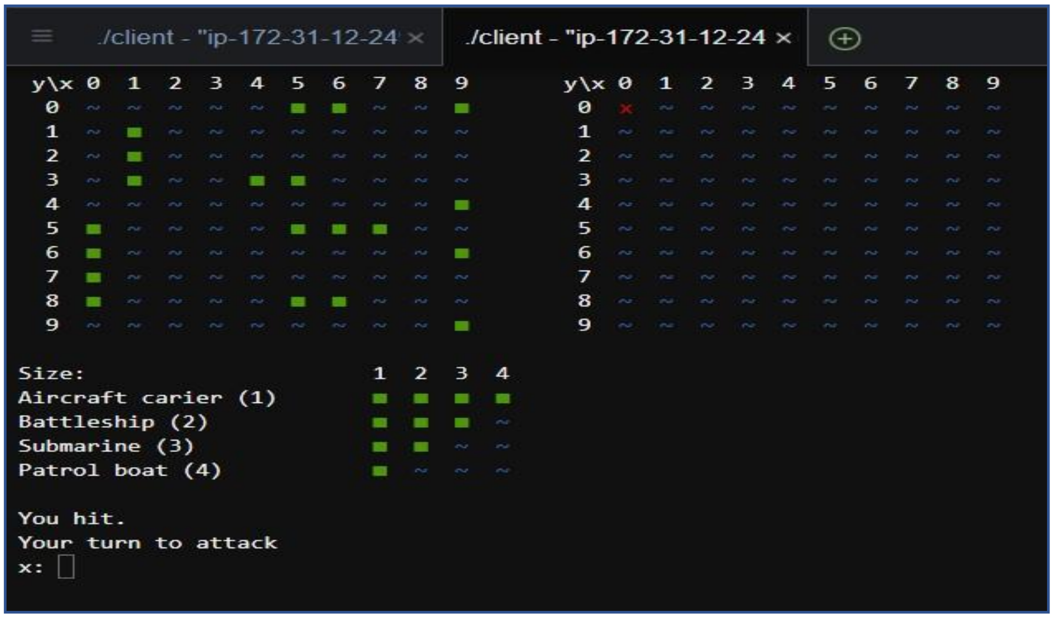
- Players can apply same method of adding the ships into the board.
- Player 2 need not wait till Player 1 is ready, and instead can simultaneously create the Placing board.

3. Start to attack



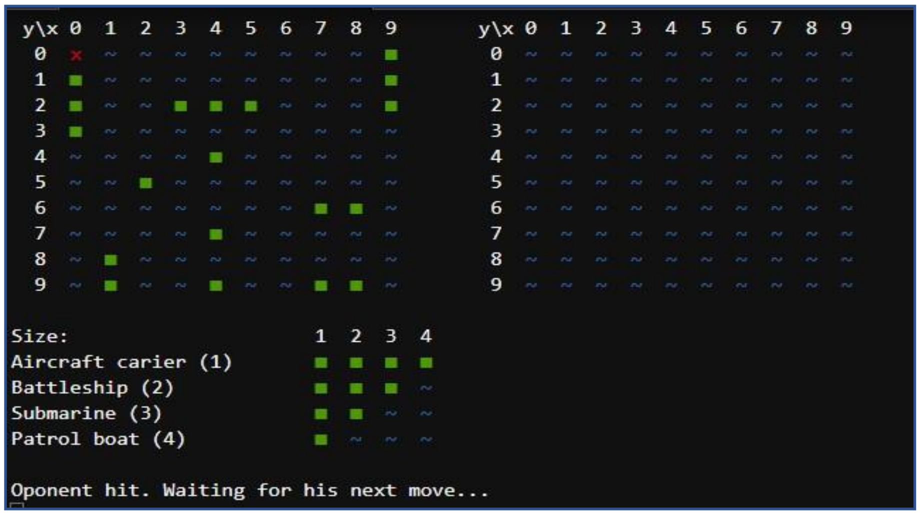
```
./client - "ip-172-31-12-24 ×    ./client - "ip-172-31-12-24 ×

y\x 0  1  2  3  4  5  6  7  8  9       y\x 0  1  2  3  4  5  6  7  8  9
 0  ~  ~  ~  ~  ~  ■  ■  ~  ~  ■        0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 1  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 2  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 3  ~  ■  ~  ~  ■  ■  ~  ~  ~  ~        3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ■        4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 5  ■  ~  ~  ~  ~  ■  ■  ■  ~  ~        5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 6  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■        6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 7  ■  ~  ~  ~  ~  ~  ~  ~  ~  ~        7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 8  ■  ~  ~  ~  ~  ■  ■  ~  ~  ~        8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ■        9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                       1  2  3  4
Aircraft carier (1)         ■  ■  ■  ■
Battleship (2)              ■  ■  ■  ~
Submarine (3)               ■  ■  ~  ~
Patrol boat (4)             ■  ~  ~  ~

Your turn to attack
x: 0
y: 0
```

Player 2 puts the missile at (0,0)



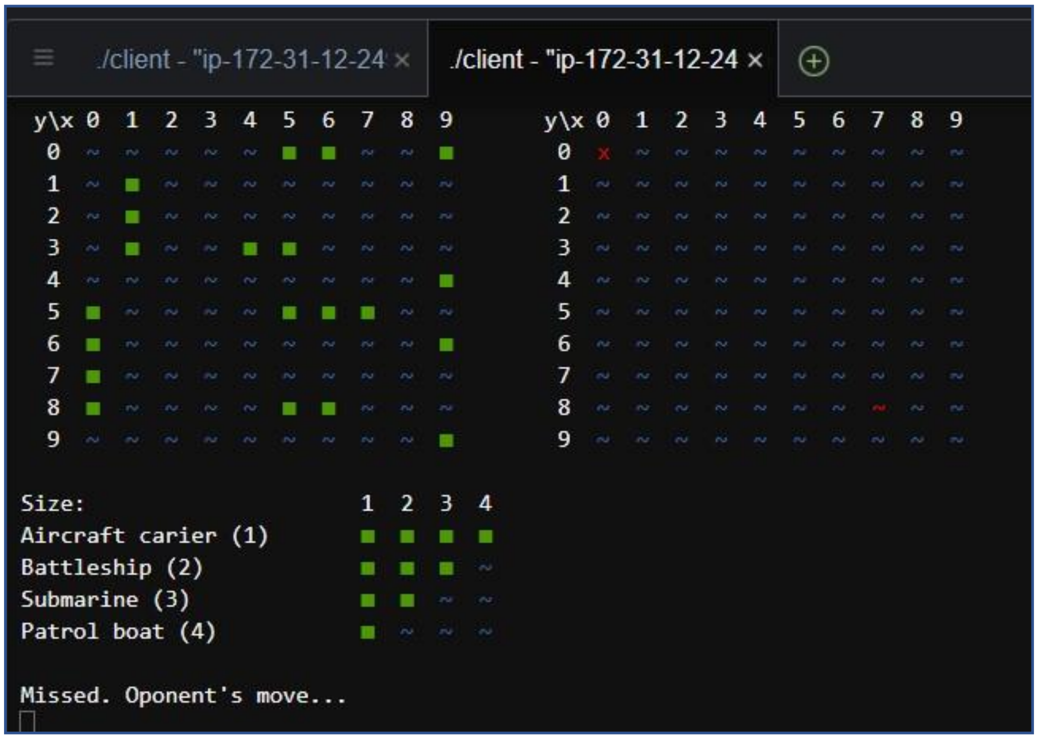
```
./client - "ip-172-31-12-24 ×    ./client - "ip-172-31-12-24 ×

y\x 0  1  2  3  4  5  6  7  8  9       y\x 0  1  2  3  4  5  6  7  8  9
 0  ~  ~  ~  ~  ~  ■  ■  ~  ~  ■        0  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
 1  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 2  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 3  ~  ■  ~  ~  ■  ■  ~  ~  ~  ~        3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ■        4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 5  ■  ~  ~  ~  ~  ■  ■  ■  ~  ~        5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 6  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■        6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 7  ■  ~  ~  ~  ~  ~  ~  ~  ~  ~        7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 8  ■  ~  ~  ~  ~  ■  ■  ~  ~  ~        8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ■        9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                       1  2  3  4
Aircraft carier (1)         ■  ■  ■  ■
Battleship (2)              ■  ■  ■  ~
Submarine (3)               ■  ■  ~  ~
Patrol boat (4)             ■  ~  ~  ~

You hit.
Your turn to attack
x:
```

Since there was a ship at (0,0) It registered as **hit** and the coordinate turns into x
(PLAYER 2 POV)
Since he hit ,he has one more turn to shoot.



```
y\x 0  1  2  3  4  5  6  7  8  9       y\x 0  1  2  3  4  5  6  7  8  9
 0  x  ~  ~  ~  ~  ~  ~  ~  ~  ■        0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 1  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■        1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 2  ■  ~  ~  ■  ■  ■  ~  ~  ~  ■        2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 3  ■  ~  ~  ~  ~  ~  ~  ~  ~  ~        3  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 4  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~        4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 5  ~  ~  ■  ~  ~  ~  ~  ~  ~  ~        5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 6  ~  ~  ~  ~  ~  ~  ~  ■  ■  ~        6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 7  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~        7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 8  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~        8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
 9  ~  ■  ~  ~  ■  ~  ~  ■  ■  ~        9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                       1  2  3  4
Aircraft carier (1)         ■  ■  ■  ■
Battleship (2)              ■  ■  ■  ~
Submarine (3)               ■  ■  ~  ~
Patrol boat (4)             ■  ~  ~  ~

Oponent hit. Waiting for his next move...
```

Hit was registered in Player 1 board also and that coordinate was also changed into x
(Player 1 POV)

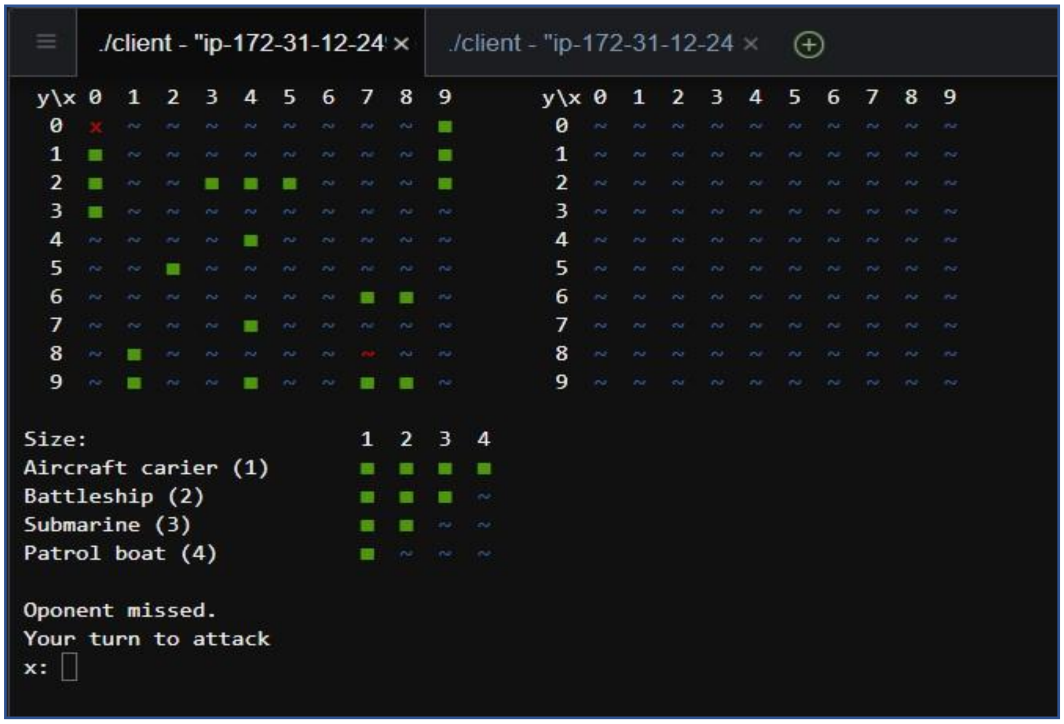


```
./client - "ip-172-31-12-24 ×   ./client - "ip-172-31-12-24 ×

y\x 0 1 2 3 4 5 6 7 8 9        y\x 0 1 2 3 4 5 6 7 8 9
 0  ~ ~ ~ ~ ~ ■ ■ ~ ~ ■         0  x ~ ~ ~ ~ ~ ~ ~ ~ ~
 1  ~ ■ ~ ~ ~ ~ ~ ~ ~ ~         1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 2  ~ ■ ~ ~ ~ ~ ~ ~ ~ ~         2  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 3  ~ ■ ~ ~ ■ ■ ~ ~ ~ ~         3  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ■         4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 5  ■ ~ ~ ~ ~ ■ ■ ■ ~ ~         5  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 6  ■ ~ ~ ~ ~ ~ ~ ~ ~ ■         6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 7  ■ ~ ~ ~ ~ ~ ~ ~ ~ ~         7  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 8  ■ ~ ~ ~ ~ ■ ■ ~ ~ ~         8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ■         9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Size:                  1 2 3 4
Aircraft carier (1)    ■ ■ ■ ■
Battleship (2)         ■ ■ ■ ~
Submarine (3)          ■ ■ ~ ~
Patrol boat (4)        ■ ~ ~ ~

Missed. Oponent's move...
```

Player 2 puts the next missile at (7,8)
Since there was a ship at (7,8) It registered as **miss** and the coordinate turns into

~

(PLAYER 2 POV)
The next move goes to Player 1.

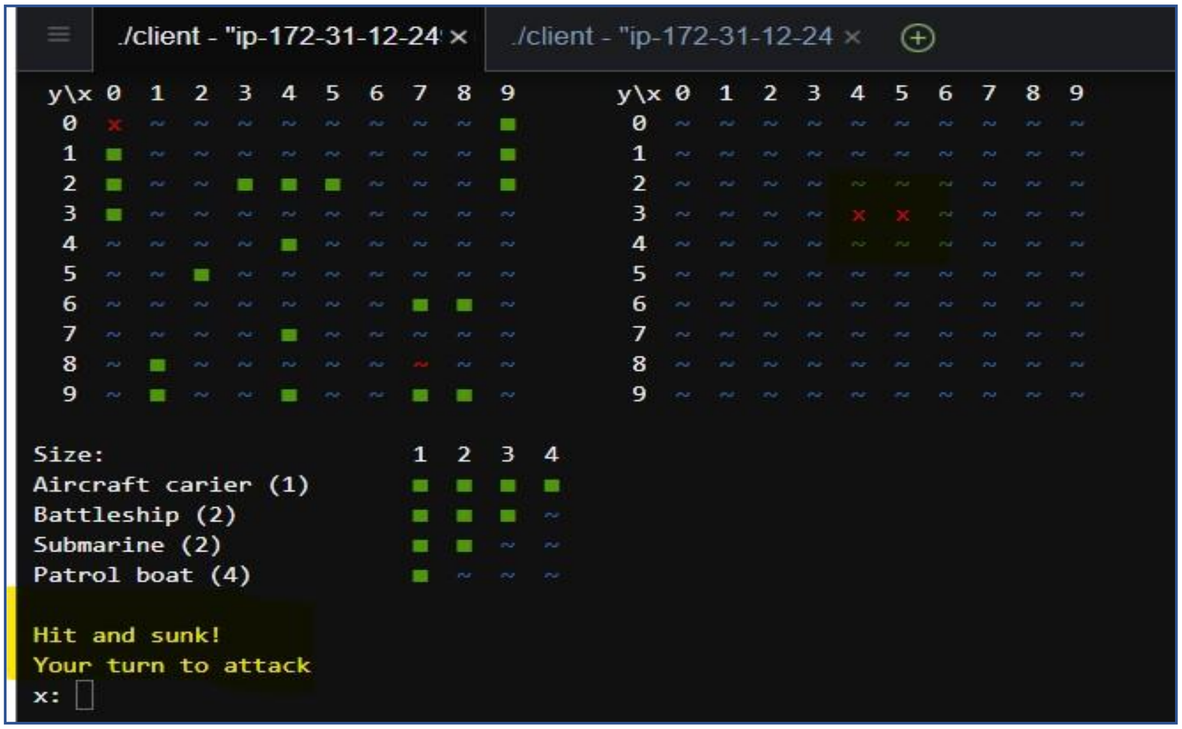


```
./client - "ip-172-31-12-24 ×   ./client - "ip-172-31-12-24 ×

y\x 0 1 2 3 4 5 6 7 8 9        y\x 0 1 2 3 4 5 6 7 8 9
 0  x ~ ~ ~ ~ ~ ~ ~ ~ ■         0  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 1  ■ ~ ~ ~ ~ ~ ~ ~ ~ ■         1  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 2  ■ ~ ~ ■ ■ ■ ~ ~ ~ ■         2  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 3  ■ ~ ~ ~ ~ ~ ~ ~ ~ ~         3  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 4  ~ ~ ~ ~ ■ ~ ~ ~ ~ ~         4  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 5  ~ ~ ■ ~ ~ ~ ~ ~ ~ ~         5  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 6  ~ ~ ~ ~ ~ ~ ~ ■ ■ ~         6  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 7  ~ ~ ~ ~ ■ ~ ~ ~ ~ ~         7  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 8  ~ ■ ~ ~ ~ ~ ~ ~ ~ ~         8  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 9  ~ ■ ~ ~ ■ ~ ~ ■ ■ ~         9  ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Size:                  1 2 3 4
Aircraft carier (1)    ■ ■ ■ ■
Battleship (2)         ■ ■ ■ ~
Submarine (3)          ■ ■ ~ ~
Patrol boat (4)        ■ ~ ~ ~

Oponent missed.
Your turn to attack
x:
```

Miss was registered in Player 1 board also and that coordinate was also changed into ~

(Player 1 POV)
Next is turn of Player 1.

This Process carries on until all the ships are sunk.

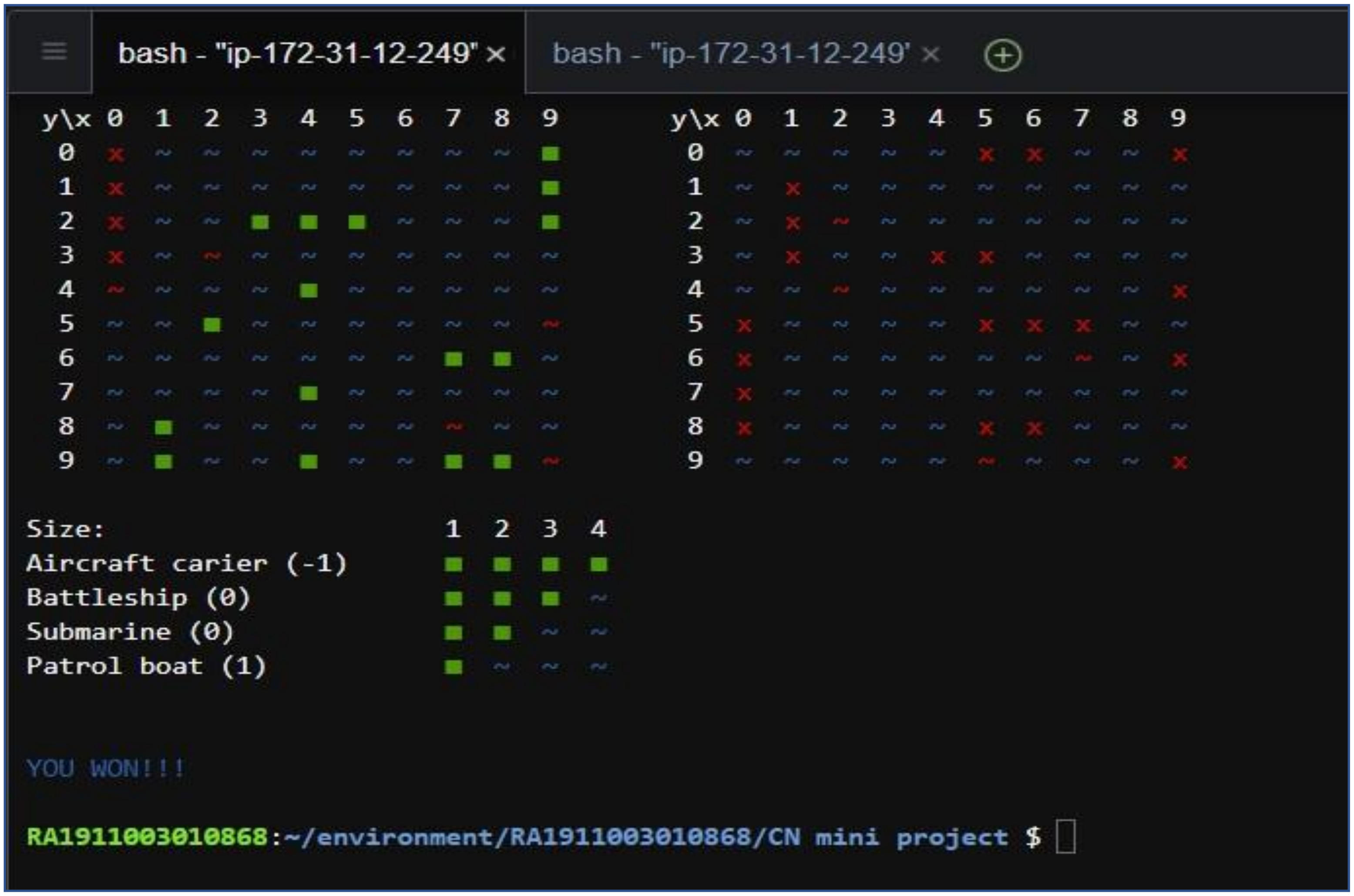


./client - "ip-172-31-12-24 ×   ./client - "ip-172-31-12-24 ×

```
 y\x 0  1  2  3  4  5  6  7  8  9        y\x 0  1  2  3  4  5  6  7  8  9
  0  x  ~  ~  ~  ~  ~  ~  ~  ~  ■          0  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  1  ■  ~  ~  ~  ~  ~  ~  ~  ~  ■          1  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  2  ■  ~  ~  ■  ■  ■  ~  ~  ~  ■          2  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  3  ■  ~  ~  ~  ~  ~  ~  ~  ~  ~          3  ~  ~  ~  ~  x  x  ~  ~  ~  ~
  4  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~          4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  5  ~  ~  ■  ~  ~  ~  ~  ~  ~  ~          5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  6  ~  ~  ~  ~  ~  ~  ~  ■  ■  ~          6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  7  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~          7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~          8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  9  ~  ■  ~  ~  ■  ~  ~  ■  ■  ~          9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                    1  2  3  4
Aircraft carier (1)      ■  ■  ■  ■
Battleship (2)           ■  ■  ■  ~
Submarine (2)            ■  ■  ~  ~
Patrol boat (4)          ■  ~  ~  ~

Hit and sunk!
Your turn to attack
x:
```

Hit and sunk is registered when an entire is ship is destroyed. In this case Player 1 hit (4,3) (5,3) sunk Player 2’s Battleship.

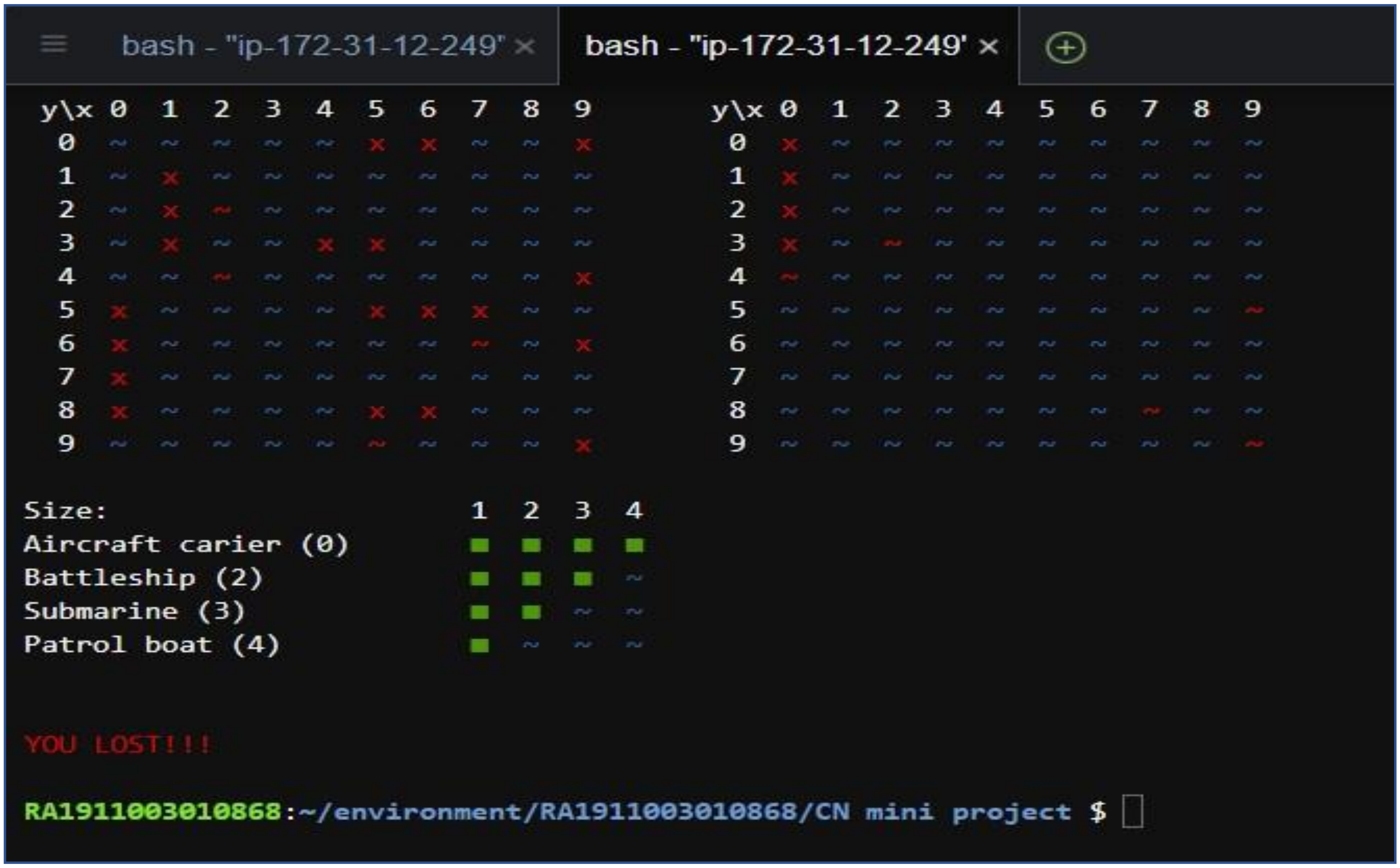
Output when **Player wins**:



bash - "ip-172-31-12-249" ×   bash - "ip-172-31-12-249' ×

```
 y\x 0  1  2  3  4  5  6  7  8  9        y\x 0  1  2  3  4  5  6  7  8  9
  0  x  ~  ~  ~  ~  ~  ~  ~  ~  ■         0  ~  ~  ~  ~  ~  x  x  ~  ~  x
  1  x  ~  ~  ~  ~  ~  ~  ~  ~  ■         1  ~  x  ~  ~  ~  ~  ~  ~  ~  ~
  2  x  ~  ~  ■  ■  ■  ~  ~  ~  ■         2  ~  x  ~  ~  ~  ~  ~  ~  ~  ~
  3  x  ~  ~  ~  ~  ~  ~  ~  ~  ~         3  ~  x  ~  ~  x  x  ~  ~  ~  ~
  4  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~         4  ~  ~  ~  ~  ~  ~  ~  ~  ~  x
  5  ~  ~  ■  ~  ~  ~  ~  ~  ~  ~         5  x  ~  ~  ~  ~  x  x  x  ~  ~
  6  ~  ~  ~  ~  ~  ~  ~  ■  ■  ~         6  x  ~  ~  ~  ~  ~  ~  ~  ~  x
  7  ~  ~  ~  ~  ■  ~  ~  ~  ~  ~         7  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  ~  ■  ~  ~  ~  ~  ~  ~  ~  ~         8  x  ~  ~  ~  ~  x  x  ~  ~  ~
  9  ~  ■  ~  ~  ■  ~  ~  ■  ■  ~         9  ~  ~  ~  ~  ~  ~  ~  ~  ~  x

Size:                         1  2  3  4
Aircraft carier (-1)          ■  ■  ■  ■
Battleship (0)                ■  ■  ■  ~
Submarine (0)                 ■  ■  ~  ~
Patrol boat (1)               ■  ~  ~  ~


YOU WON!!!

RA1911003010868:~/environment/RA1911003010868/CN mini project $ 
```

Output when **Player loses**:



```
≡   bash - "ip-172-31-12-249" ×   bash - "ip-172-31-12-249' ×   ⊕

 y\x 0  1  2  3  4  5  6  7  8  9          y\x 0  1  2  3  4  5  6  7  8  9
  0  ~  ~  ~  ~  ~  x  x  ~  ~  x            0  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
  1  ~  x  ~  ~  ~  ~  ~  ~  ~  ~            1  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
  2  ~  x  ~  ~  ~  ~  ~  ~  ~  ~            2  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
  3  ~  x  ~  ~  x  x  ~  ~  ~  ~            3  x  ~  ~  ~  ~  ~  ~  ~  ~  ~
  4  ~  ~  ~  ~  ~  ~  ~  ~  ~  x            4  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  5  x  ~  ~  ~  ~  x  x  x  ~  ~            5  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  6  x  ~  ~  ~  ~  ~  ~  ~  ~  x            6  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  7  x  ~  ~  ~  ~  ~  ~  ~  ~  ~            7  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  8  x  ~  ~  ~  ~  x  x  ~  ~  ~            8  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~
  9  ~  ~  ~  ~  ~  ~  ~  ~  ~  x            9  ~  ~  ~  ~  ~  ~  ~  ~  ~  ~

Size:                     1  2  3  4
Aircraft carier (0)       ■  ■  ■  ■
Battleship (2)            ■  ■  ■  ~
Submarine (3)             ■  ■  ~  ~
Patrol boat (4)           ■  ~  ~  ~


YOU LOST!!!

RA1911003010868:~/environment/RA1911003010868/CN mini project $ 
```

6.2. RESULT ANALYSIS

The Overall performance of the program was good. There was no delay or lag when connecting to multiple clients. The user interface was friendly and the client and server process started without any errors.
We successfully implemented a Battleship game, which required multiple clients and one server with a positive response.
The game was designed and prepared seeing the interest of modern times.
We defined whole program security by specifying that a referee must behave identically to a model implementation from the players' vantage points, and defined security of aplayer interface against a malicious opponent using the real/ideal paradigm.

This project has been able to create a new opening and the basic structure of a good design.
We believe there is a great need for more case studies in language-based security. The balance between theory (e.g., non-interference results) and practice seems heavily on the theory side. Actual case studies—warts and all—are a crucial way of grounding the field.
We were able to understand the working of Daemon processes while using TCP/IP connection, how to incorporate external libraries and how to establish concurrency.

Working in refurnishing the classics will also help us expand the experience that these games provide.

6.3. CONCLUSION & FUTURE WORK

We have used TCP/IP for this project. The benefits of using TCP/IP are good failure recovery, the ability to add networks without interrupting existing services, high error-rate handling, platform independence, and low data overhead. We have successfully implemented Battleship game. We must explore various other methods to incorporate multiple iterations of this iterations of the game. Each of which will have different rules to be followed. We can expand this code to various other game applications too.

- Menu option which lists out other versions of this game with different rules.
- Availability of database to recognize previous username and store records.
- GUI application of the Game as well as a web-application for this game.
- Bot mode, where the client can play against the computer.

7. REFERENCES

BOOKS USED

Anon. (2004). Operating Systems Fundamentals; Windows 9X Family. MarcraftInternational corporation. NIIT

Comer E. D, Stevens L. D. (1999). Internetworking With TCP/IP, Vol. II: Design,Implementing,And Internals. 3rd Ed. Prentice-Hall of India Private Limited

Groth D, Bergersen B, Catura-HouserT. (1999).

Networking+.SYBEX Inc

EBOOKS USED

Lowe D.(2005). Networking for Dummies. 7th Ed. [eBook]. Wiley Publishing, Inc. Available from : http://www.dummiesbooks.net/Networking-dummies-books.html. [cited 01 October 2009]

Lowe D.(2005). Networking All-In-One Desk Reference for Dummies. 2nd Ed.[eBook]. Wiley Publishing, Inc. Available from : http://www.dummiesbooks.net/Networking-dummies-books.html. [cited 01 October 2009]

WEBSITES USED

Kanye R. What is the difference between a lan and the internet?. [online]. [Http://www.wisegeek.com/what-is-the-difference-between-a-lan-and-the-internet. htm]. 2009. [cited 01 October 2009]

Parker T, Siyan S. K. Overview of TCP/IP. [online]. [Http://www.informit.com/articles/article.aspx?P=28782]. 2002. [cited 01 October 2009].

Beej's Guide to Network Programming: using Internet Sockets[online] [https://beej.us/guide/bgnet/]