Server

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#define ROW 10

#define COL 10

char maze[10][10];

int player\_row = 0;

int player\_col = 0;

int cnt=0;

void create\_maze() {

// Initialize the maze

player\_row=0;

player\_col=0;

int i,j;

for ( i = 0; i < ROW; i++) {

for(j = 0; j < COL; j++) {

maze[i][j] = ' ';

}

}

maze[0][0] = 'S';

maze[ROW - 1][COL - 1] = 'E';

// Place traps in the maze

for (i = 0; i < 10; i++) {

int x = rand() % ROW;

int y = rand() % COL;

maze[x][y] = 'X';

}

maze[ROW-2][COL-1]=' ';

maze[ROW-1][COL-2]=' ';

}

void display\_maze(int client\_socket) {

char buffer[1024];

memset(buffer, 0, sizeof(buffer));

int i,j;

printf("\n===============MAZE===============\n\n");

for (i = 0; i < ROW; i++) {

for (j = 0; j < COL; j++) {

printf("%c |",maze[i][j]);

sprintf(buffer, "%s%c ", buffer, maze[i][j]);

}

printf("\n------------------------------\n");

strcat(buffer, "\n");

}

send(client\_socket, buffer, sizeof(buffer), 0);

}

int main() {

create\_maze();

// Create a socket

int server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

perror("Socket creation failed");

exit(1);

}

// Bind the socket to an address and port

struct sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(12345);

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(server\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)) < 0) {

perror("Binding failed");

exit(1);

}

// Listen for incoming connections

if (listen(server\_socket, 5) == 0) {

printf("Listening for clients...\n");

} else {

perror("Listening failed");

exit(1);

}

// Accept a client connection

int client\_socket;

struct sockaddr\_in client\_address;

socklen\_t client\_address\_len = sizeof(client\_address);

while(cnt<10){

client\_socket = accept(server\_socket, (struct sockaddr\*)&client\_address, &client\_address\_len);

if (client\_socket < 0) {

perror("Acceptance of client failed");

exit(1);

}

printf("Client - %d connected\n",++cnt);

while (1) {

display\_maze(client\_socket);

char move;

recv(client\_socket, &move, sizeof(move), 0);

// Update player's position based on the move

if (move == 'w' && player\_row > 0 && maze[player\_row - 1][player\_col] != 'X') {

player\_row--;

} else if (move == 's' && player\_row < ROW - 1 && maze[player\_row + 1][player\_col] != 'X') {

player\_row++;

} else if (move == 'a' && player\_col > 0 && maze[player\_row][player\_col - 1] != 'X') {

player\_col--;

} else if (move == 'd' && player\_col < COL - 1 && maze[player\_row][player\_col + 1] != 'X') {

player\_col++;

}

// Check if the player has reached the end of the maze

if (maze[player\_row][player\_col] == 'E') {

display\_maze(client\_socket);

printf("Congratulations, you won!\n\n\n");

char a[10]="exit";

send(client\_socket,a,sizeof(a),0);

create\_maze();

break;

}

maze[player\_row][player\_col] = 'P';

}

close(client\_socket);

}

printf("\n\nServer Closes.....\n");

close(server\_socket);

return 0;

}