```c
/**
 * fifteen.c
 *
 * CS50 AP
 * Name: Andres Joaquin Bolanos Chang
 *
 * Implements Game of Fifteen (generalized to d x d).
 *
 * Usage: fifteen d
 *
 * whereby the board's dimensions are to be d x d,
 * where d must be in [DIM_MIN,DIM_MAX]
 *
 * Note that usleep is obsolete, but it offers more granularity than
 * sleep and is simpler to use than nanosleep; `man usleep` for more.
 */

// necessary for usleep
#define _XOPEN_SOURCE 500

// libraries to include
#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>

// constants
#define DIM_MIN 3
#define DIM_MAX 9

// globally declared board
int board[DIM_MAX][DIM_MAX];

// globally declared board dimension
int d;
int blankspace;
int x;
int y;

// prototypes
void clear(void);
void greet(void);
void init(void);
void draw(void);
bool move(int tile);
bool won(void);
```

```c

int main(int argc, string argv[])
{
    // Todo 00: Si hay un numero de comentarios diferente a 2, salirse
    if (argc != 2)
    {
        printf("Usage: fifteen d\n");
        return 1;
    }

    // TODO 01: la dimension en el arg1 deben estar entre MIN y MAX
    d = atoi(argv[1]);
    if (d < DIM_MIN || d > DIM_MAX)
    {
        printf("Board must be between %i x %i and %i x %i, inclusive.\n",
            DIM_MIN, DIM_MIN, DIM_MAX, DIM_MAX);
        return 2;
    }

    x = (d - 1);
    y = (d - 1);

    // open log file to record moves
    FILE* file = fopen("log.txt", "w");
    if (file == NULL)
    {
        return 3;
    }

    // TODO 02: El juego te saluda
    greet();

    // TODO 03: inicializar el "board"
    init();

    // accept moves until game is won
    while (true)
    {
        // TODO 04: Borra/limpia el terminal
        clear();

        // TODO 05: Dibuja el "board"
        draw();

        for (int i = 0; i < d; i++)
        {
            for (int j = 0; j < d; j++)
            {
                fprintf(file, "%i", board[i][j]);
                if (j < d - 1)
                {
                    fprintf(file, "|");
                }
            }
            fprintf(file, "\n");
        }
        fflush(file);

        // TODO 06: Checa si ya ganaste
        if (won())
        {
            printf("ftw!\n");
            break;
        }

        // TODO 07: Indicar cual "tile" mover
        printf("Tile to move: ");
        int tile = GetInt();

        if (tile == 0)
        {
            break;
        }

        fprintf(file, "%i\n", tile);
        fflush(file);

        // TODO 08: Si no se puede mover, te dice que es ilegal
        if (!move(tile))
        {
            printf("\nIllegal move.\n");
            usleep(500000);
        }

        // TODO 09: Hace que se "duerma" (suspenda) microsegundos el programa
        usleep(500000);
    }

    // close log
    fclose(file);

    // TODO 10: Termina el programa
    return 0;
}

/**
```

```c
 * Clears screen using ANSI escape sequences.
 */
void clear(void)
{
    printf("\033[2J");
    printf("\033[%d;%dH", 0, 0);
}

/**
 * Greets player.
 */
void greet(void)
{
    clear();
    printf("WELCOME TO GAME OF FIFTEEN\n");
    usleep(2000000);
}

/**
 * Initializes the game's board with tiles numbered 1 through d*d - 1
 * (i.e., fills 2D array with values but does not actually print them).
 */
void init(void)
{
    int spacenum = (d * d) - 1;
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            board[i][j] = spacenum;
            spacenum = spacenum - 1;
        }
    }
    if ((d % 2) == 0)
    {
        int temp = board[d - 1][d - 2];
        board[d - 1][d - 2] = board[d - 1][d - 3];
        board[d - 1][d - 3] = temp;
    }
}

/**
 * Prints the board in its current state.
 */
void draw(void)
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            if (board[i][j] < 10)
            {
                if (board[i][j] == 0)
                {
                    printf(" _ ");
                }
                else
                {
                    printf(" %d ", board[i][j]);
                }
            }
            else
            {
                printf("%d ", board[i][j]);
            }
        }
        printf("\n\n");
    }
}

/**
 * If tile borders empty space, moves tile and returns true, else
 * returns false.
 */
bool move(int tile)
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            if (tile == board[i][j])
            {
                int blankspace = 0;

                if (((x == (i - 1)) && (j == y)) || ((i == x) && (y == (j + 1))
                ) || ((i == x) && (y == (j - 1))) || ((x == (i + 1)) && (j == y)
                ))
                {
                    board[x][y] = tile;
                    board[i][j] = blankspace;
                    x = i;
                    y = j;
                    printf("\n");
                    return true;
                }
            }
        }
    }
    return false;
}

/**
 * Returns true if game is won (i.e., board is in winning configuration),
 * else false.
 */
bool won(void)
{
    int n = 1;
    // iterates through board
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            // if any tile != counter, counting from 0, return false
            if (board[i][j] == n)
            {
                n++;
                if (n == (d * d) && board[d - 1][d - 1] == 0)
                {
                    return true;
                }
            }
        }
    }
    return  false;
}
```