```c
/**
 * fifteen.c
 *
 * CS50 AP
 * Name: Alexandra Pruenda
 *
 * 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 * Note that usleep is obsolete, but it offers more granularity than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>

// constants
#define DIM_MIN 3
#define DIM_MAX 9

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

// globally declared board dimension
int dim;

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

int main(int argc, string argv[])
{
    // 0. TODO Indicar el numero de argumentos que puede introducir
    if (argc != 2)
```

```
    {
        printf("Usage: fifteen dimension\n");
        return 1;
    }

    // 1. TODO Dimensiones validas
    dim = atoi(argv[1]);
    if (dim < DIM_MIN || dim > 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;
    }

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

    // 2. TODO El usuario
    greet();

    // 3. TODO Comenzar el cuadro
    init();

    // accept moves until game is won
    while (true)
    {
        // 4. TODO Limpiar la pantalla
        clear();

        // 5. TODO Dibujar como esta el cuadro
        draw();

        // log the current state of the board (for testing)
        for (int i = 0; i < dim; i++)
        {
            for (int j = 0; j < dim; j++)
            {
                fprintf(file, "%i", board[i][j]);
                if (j < dim - 1)
                {
                    fprintf(file, "|");
                }
            }
            fprintf(file, "\n");
```

```c
        }
        fflush(file);

        // 6. TODO Revisar si gana
        if (won())
        {
            printf("ftw!\n");
            break;
        }

        // 7. TODO Empezar a moverse
        printf("Tile to move: ");
        int tile = GetInt();

        // quit if user inputs 0 (for testing)
        if (tile == 0)
        {
            break;
        }

        // log move (for testing)
        fprintf(file, "%i\n", tile);
        fflush(file);

        // 8. TODO Moverse si es posible, sino decir que es un "illegal move"
        if (!move(tile))
        {
            printf("\nIllegal move.\n");
            usleep(500000);
        }

        // 9. TODO Reposar por el bien del juego
        usleep(500000);
    }

    // close log
    fclose(file);

    // 10.TODO Termina todo
    return 0;
}

/**
 * Clears screen using ANSI escape sequences.
 */
void clear(void)
{
    printf("\033[2J");
```

```c
    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)
{
    // TODO
    // Los numeros del cuadro empezando en 0
    int lastnum = dim * dim - 1;
    for (int i = 0; i < dim; i++)
    {
        for (int j = 0; j < dim; j++)
        {
            board[i][j] = lastnum;
            lastnum--;
        }
    }
    // Cambiar siempre el dos y el uno
    if ((dim * dim - 1) % 2 != 0)
    {
        // El valor original de 2 que sea 1
        board[dim - 1][dim - 2] = 2;
        // El valor original de 1 que sea 2
        board[dim - 1][dim - 3] = 1;
    }
}

/**
 * Prints the board in its current state.
 */
void draw(void)
{
    // TODO
    // Filas
    for(int i = 0; i < dim; i++)
```

```c
    {
        // Columnas
        for (int j = 0; j < dim; j++)
        {
            if (board[i][j] == 0)
            {
                printf("  _ ");
            }

            else
            {
                printf("%3d ", board[i][j]);
            }
        }
        printf("\n");
    }
}
/**
 * If tile borders empty space, moves tile and returns true, else
 * returns false.
 */
bool move(int tile)
{
    for (int i = 0; i < dim; i++)
    {
        // Columnas del cuadro
        for (int j = 0; j < dim; j++)
        {
            if (board[i][j] == tile)
            {
                if ((i + 1 < dim) && (board[i + 1][j] == 0))
                {
                    board[i + 1][j] = tile;
                    board[i][j] = 0;
                }
                else if ((i - 1 >= 0) && (board[i - 1][j] == 0))
                {
                    board[i - 1][j] = tile;
                    board[i][j] = 0;
                }
                else if ((j + 1 < dim) && (board[i][j + 1] == 0))
                {
                    board[i][j + 1] = tile;
                    board[i][j] = 0;
                }
                else if ((j - 1 >= 0) && (board[i][j - 1] == 0))
                {
                    board[i][j - 1] = tile;
```

```
                    board[i][j] = 0;
                }
                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;
    for (int i = 0; i < dim; i++)
    {
        for(int j = 0; j < dim; j++)
        {
            if (board[i][j] == n)
            {
                n++;
                if ((n == (dim * dim)) && (board[dim - 1][dim - 1] == 0))
                {
                    return true;
                }
            }
        }
    }
    return false;
}
```