INSTITUTO TECNOLÓGICO DE COSTA RICA

Campus Tecnológico Central Cartago
IC-3101 Arquitectura de Computadores
Estudiante 1: Allan Bolaños Barrientos
Número de carné 2024145458
Profesor: Ms.c Esteban Arias Mendez

Ingeniería en Computación
II Semestre 2024
Estudiante 2: Brian Ramírez Arias
Número de carné: 2024109557

Proyecto 2 — Super TIC-TAC-TOE

Fecha de entrega: 11/30/24

Abstract

This documentation explains the operation of the program designed in C and Assembly, which consists of a game known as TIC TAC TOE, however this is a little more complex since it has mini-games in each space of the main game. Graphics were programmed in C using the GTK library, the memory matrix and movements were managed in NASM.

Contents

1 Descripción del programa.

El proyecto esta diseñado para aumentar la complejidad y las posibilidades tácticas del juego tradicional. Este proyecto consiste en la creación de un tablero principal compuesto por 9 celdas (3x3), donde cada celda, a su vez, contiene un tablero individual de Tic-Tac-Toe más pequeño.

La mecánica principal se basa en que cada jugador, al ganar uno de los tableros pequeños, domina la celda correspondiente del tablero principal. El objetivo final es ganar el tablero principal formando una línea de tres celdas controladas horizontal, vertical o diagonal. Tambien se debe tener encuenta que el movimiento realizado en un tablero pequeño determina el próximo tablero donde jugará el oponente. Por ejemplo, si un jugador coloca su ficha en la esquina inferior derecha de un tablero pequeño, el siguiente turno deberá jugarse en el tablero pequeño ubicado en esa misma esquina del tablero principal pero si el tablero correspondiente ya ha sido ganado o está lleno, el jugador tiene libertad de elegir cualquier otro tablero habilitado.

El lenguaje C, junto con la biblioteca GTK, se utiliza para gestionar la interfaz gráfica del juego. Esto incluye el diseño visual del tablero, la interacción con los usuarios y la visualización de los movimientos realizados por cada jugador. [1][2][3]

El ensamblador se utiliza para las operaciones críticas del juego. Entre las funciones implementadas en NASM se encuentran.

- Gestión de Memoria: Asignación estatica para las estructuras de datos del tablero.
- Manipulación de la Matriz: Cálculo de posiciones, control de sub-tableros y actualización del estado general del juego.
- Control de Movimientos: Verificación de la validez de los movimientos y cálculo del siguiente tablero a habilitar.
- Detección de Ganadores: Verificación de condiciones de victoria en los tableros pequeños y en el tablero principal.
- Reglas Especiales: Gestión de reglas dinámicas como la apertura de sub-tableros específicos o la habilitación global de tableros.

El proyecto combina los módulos de C y ASM a través de ligas, lo que permite que ambos lenguajes trabajen en conjunto. Las funciones en ensamblador están diseñadas para ser llamadas desde el código C, garantizando una transición fluida entre la lógica gráfica y las operaciones del juego.[4]

2 Estructura del proyecto.

El proyecto esta divido en carpetas para mayor organizacion, durante esta documentacion se detallara que se encuentra en cada una.

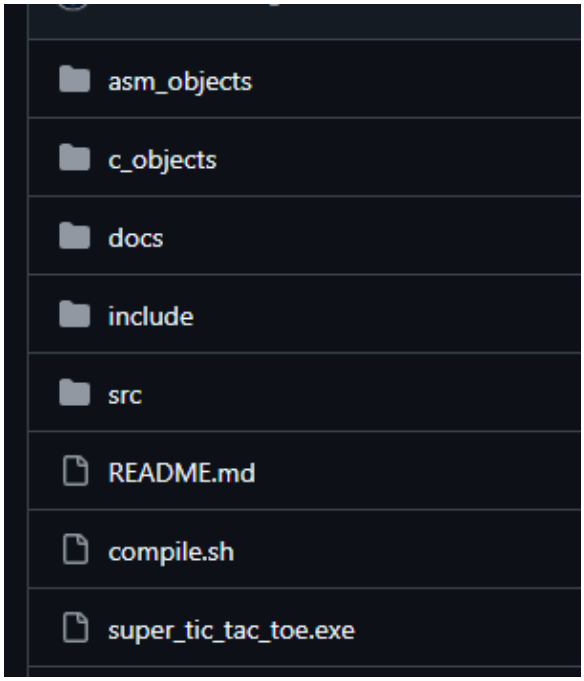

Figure 1: Carpetas generales

Como se puede ver el la figura 1, las dos primeras carpetas lo que almacenan son los objetos que se generan al compilar los archivos principales del programas, esto se logra gracias a que se especifica la ruta donde se quiere guardar el objeto a la hora de compilar como se muestra en la figura 7.

Figure 2: Headers

Como se puede ver en la figura 2, la carpeta include contiene los headers los cuales son muy necesarios para vincular funciones del programa en diferentes archivos y lenguajes ya que en cada uno se especifica las funciones que se deben recibir o tomar en cuenta desde C que son de NASM.

Figure 3: Carpeta de codigo principal

En la figura 3 se muestra las diviciones de la carpeta src, en cada una de ellas se encuetran los archivos con codigo en cada lenguaje, en la carpeta interface hay un archivo en C para llamar a la grafica contruida en glade.

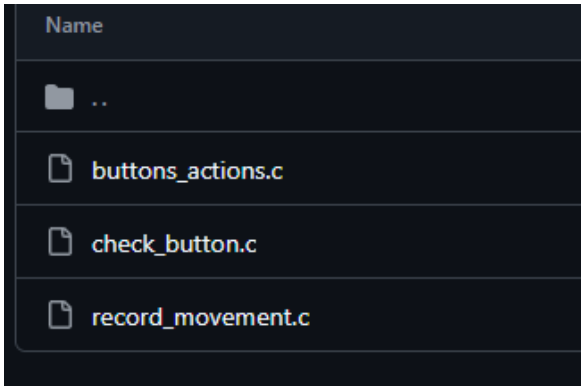

Figure 4: Carpeta de funciones en C

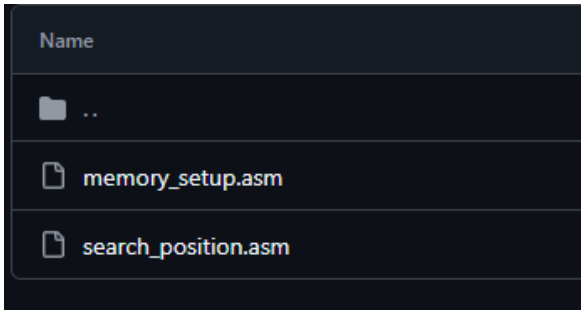

Figure 5: Carpeta de funciones en NASM

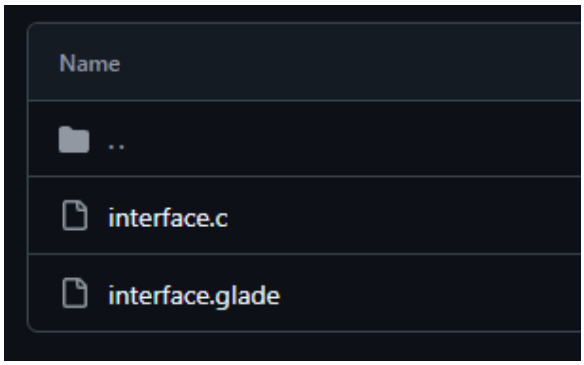

Figure 6: Carpeta de llamada y archivo de la interfaz

Es muy importante mencionar el archivo compile.sh, el cual consiste en un script ejecutable de linux que se encarga de compilar todo y ubicar cada objeto donde debe ir, ademas genera el archivo a ejecutar.

```
#!/bin/bash

# Dependencies: gtk+-3.0
# Asegúrate de tener instaladas las dependencias
# sudo apt update
# sudo apt install libgtk-3-dev gcc g++

# Nombre del ejecutable
EXEC_NAME="super_tic_tac_toe.exe"

# Compilación de archivos NASM
nasm -f elf64 -g -F dwarf src/asm_functions/search_position.asm -o asm_objects/search_position.o
nasm -f elf64 -g -F dwarf src/asm_functions/memory_setup.asm -o asm_objects/memory_setup.o

# Archivos objeto NASM
NASM_OBJECTS="asm_objects/search_position.o asm_objects/memory_setup.o"

# Compilación de archivos C
gcc -m64 -fPIE -I include -c src/main.c -o c_objects/main.o
gcc -m64 -fPIE -I include -c src/c_functions/record_movement.c -o c_objects/record_movement.o

# Archivos objeto C
C_OBJECTS="c_objects/main.o c_objects/interface.o c_objects/buttons_actions.o c_objects/record_movement.o c_objects/check_button.o"

# Compilación de archivos dependientes de GTK
gcc -m64 -fPIE -c -I include -o c_objects/interface.o src/interface/interface.c $(pkg-config --cflags gtk+-3.0)
gcc -m64 -fPIE -c -I include -o c_objects/buttons_actions.o src/c_functions/buttons_actions.c $(pkg-config --cflags gtk+-3.0)
gcc -m64 -fPIE -c -I include -o c_objects/check_button.o src/c_functions/check_button.c $(pkg-config --cflags gtk+-3.0)

# Enlace final con PIE habilitado
gcc -m64 -fPIE -pie -o $EXEC_NAME $C_OBJECTS $NASM_OBJECTS $(pkg-config --cflags --libs gtk+-3.0)

# Ejecución
./$EXEC_NAME
```

Figure 7: Script para compilar.

3 Flujo del programa.

El programa sigue un flujo simple he intuitivo para generar las llamadas y gestinar la memoria a continuacion se detallan los procesos mas importantes.

3.1 Funcion main.

La funcion main ejecuta una funcion que elije de manera aleatoria uno o dos, si es uno significa que el primero que jugara sera el primer jugador(X) si es dos significa que juega el segundo (O), posteriormente se genera la llamada a las funciones para iniciar la memoria que se necesita en ensamblador y para iniciar la grafica.

```c
#include <interface.h>
#include <asm_utils.h>
#include <stdio.h>
#include <stdlib.h>

void select_first_player() {
    long first_player = (rand() % 2) + 1;
    main_matriz[90] = first_player;
    printf("Jugador %ld empieza\n", first_player);
}

// Main function
void main(int argc, char **argv) {

    select_first_player();

    for (int i = 0; i < 91; i++) {
        printf("%ld ", main_matriz[i]);
    }

    call_login(argc, argv);
}
```

Figure 8: Funcion principal.

3.2 Funciones de interfaz.

En la interfaz existen muchas funciones auxiliares sin embargo son tres principales, una de ellas es la primera que se llama desde el main para darle la bienvenida a los

usarios y recibir informacion de los nombre, aqui se manejan ciertas validaciones, cuando el usuario inicia el juego, esta pantalla se oculta y muestra la pantalla del juego con el tablero.

```
295  void call_login(int argc, char **argv){
296   // Initialize GTK
297      gtk_init(&argc, &argv);
298
299      // Load the interface from the glade file
300      builder = gtk_builder_new_from_file(glade_path);
301
302      if(builder == NULL){printf("Error: Could not load interface.glade\n");return;}
303      // Get the login window from the glade file
304      login_window = GTK_WIDGET(gtk_builder_get_object(builder, "login_window"));
305      gtk_window_set_title(GTK_WINDOW(login_window), "Super Tic Tac Toe");
306      if (login_window == NULL) {printf("Error: Could not find login_window in the Glade file\n");return;}
307
308      // to connect glade signals in the glade file
309      g_signal_connect(login_window, "destroy", G_CALLBACK(closeWindow), NULL);
310      gtk_builder_connect_signals(builder, NULL);
311
312      //loading the login button
313      login_button= GTK_WIDGET(gtk_builder_get_object(builder, "login_button"));
314      g_signal_connect(login_button, "clicked", G_CALLBACK(start_game), NULL);
315
316      //loading the cpu option
317      cpu_option = GTK_WIDGET(gtk_builder_get_object(builder, "CPU_option"));
318      if (cpu_option == NULL) {printf("Error: Could not find cpu_option in the Glade file\n");return;}
319      g_signal_connect(cpu_option, "notify::active", G_CALLBACK(check_cpu), NULL);
320
321
322      // glade loop
323      gtk_widget_show_all(login_window);
324      gtk_main();
325
326      //If the user close the window or don't login properly
327      if (login == FALSE) return;
328      else call_game();
329  }
```

Figure 9: Funcion login.

En caso de que el usuario elija vs CPU el segundo nombre se bloqueara y se pondrá CPU por default, además se activara una flag que indica que el segundo jugador tomara el rol de CPU, esto funciona posteriormente para saber fácilmente si es un moviendo de CPU o de usuario el que se debe esperar.

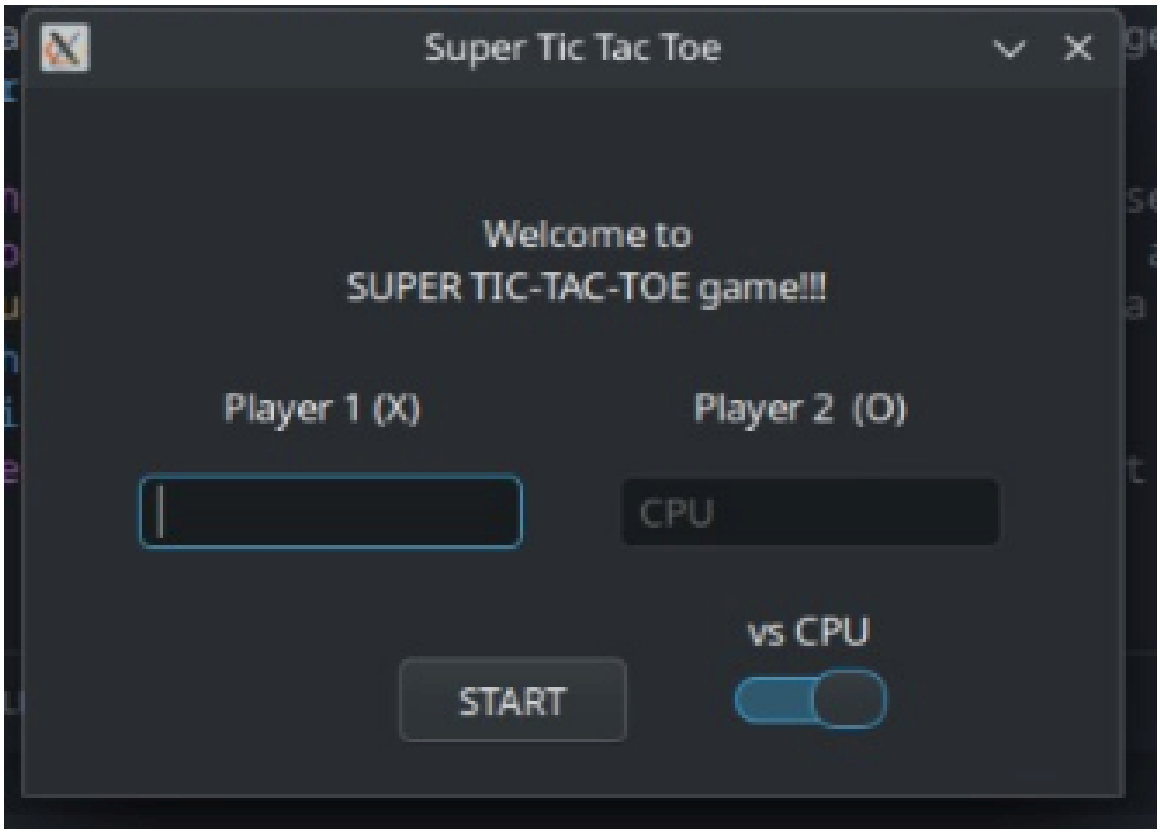

Figure 10: Login.

El botón de comienzo se le configura una acción para que cuando sea tocado, pueda llamar a la función que inicia el juego.

El durante el juego cada casilla es un botón, cada botón tiene una acción configurada para que llame a una función cada vez que se toque. Cada vez que el usuario toca una celda, ocurren muchas cosas internamente ya que se hacen muchas validaciones a nivel de Ensamblador para saber si la jugada esta disponible o no, también a nivel grafico se procura bloquear las celdas no disponibles para hacer entender al usuario donde puede jugar.

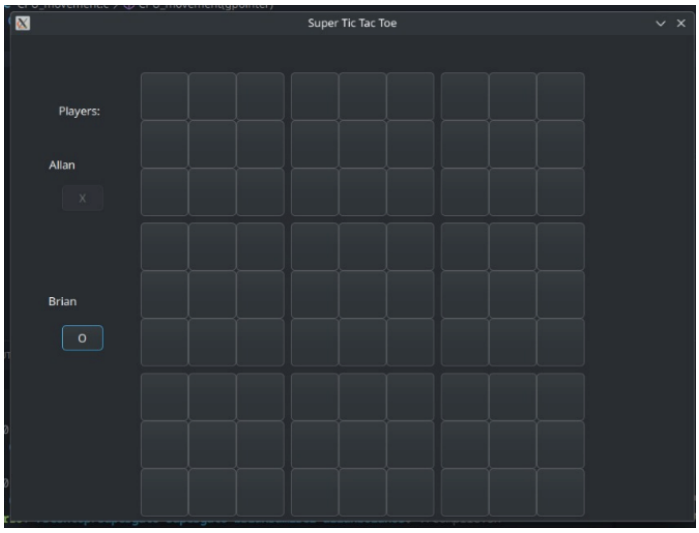

Figure 11: Inicio del juego.

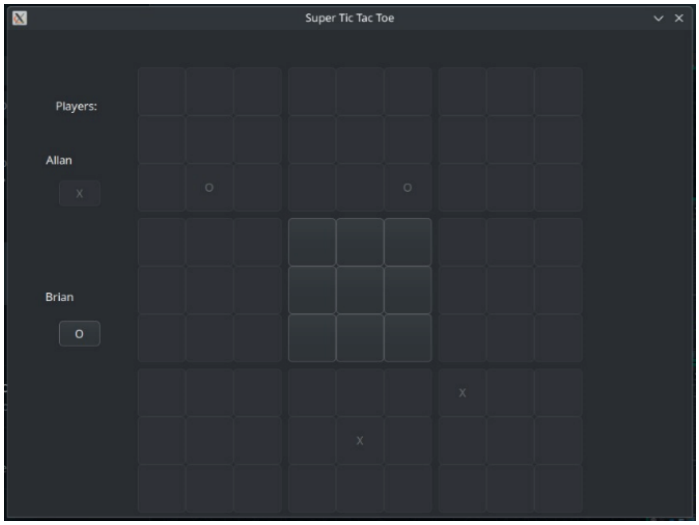

Figure 12: Durante el juego.

```
//display who plays first
displayWhoPlay();
Button *buttons[ROWS][COLS];//matriz for each space in the main_grid
// loading the buttons in the main_grid
for(int gridNum=0; gridNum < 9; gridNum++){
    for(int i = 0; i < ROWS; i++){
        for(int j = 0; j < COLS; j++){
            buttons[i][j]= malloc(sizeof(Button));
            buttons[i][j]->buttonObj= gtk_button_new_with_label("");
            buttons[i][j]->row = i;
            buttons[i][j]->col = j;
            buttons[i][j]->gridNum = gridNum;
            gtk_widget_set_size_request(buttons[i][j]->buttonObj, 70, 70);
            gtk_grid_attach(GTK_GRID(gtk_builder_get_object(builder, grid_names[gridNum])), buttons[i][j]->buttonObj, buttons[i][j]
            g_signal_connect(buttons[i][j]->buttonObj, "clicked", G_CALLBACK(button_clicked), buttons[i][j]);

        }
    }
}
```

Figure 13: Generacion de botones en pantalla.

```
4   typedef struct button{
5       GtkWidget *buttonObj;
6       int row;
7       int col;
8       int gridNum;
9   } Button;
```

Figure 14: Struct Button.

Al final cuando el código de ensamblador indica que hay un ganador, este se pone en la última posición de la matriz principal la cual es consultada desde la interfaz para refrescar la pantalla y presentar quien gano.

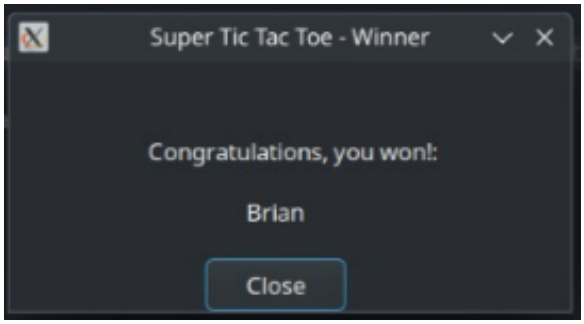

Figure 15: Pantalla del ganador.

Cuando se está jugando en modo CPU, se inicia un timer propio de GTK que cada cierto tiempo verifica si es turno del jugador 2 para ver donde se va a jugar, recordemos que el CPU siempre es el jugador 2.

```
234         // CPU action
235         if (CPUflag )
236             g_timeout_add_seconds(4, CPU_movement, builder); // Call the CPU movement e
237
```

Figure 16: Timer del CPU.

3.3 Llamadas a las funciones de NASM.

Para llamar a la función de ensamblador, lo que se hace es definir en el header asm utils las funciones como externas y se llaman cada vez que se produzca una acción. Cuando se está jugando una partida de usuario contra usuario, lo que se hace es escuchar y esperar cada botón activo en pantalla para ver que quiere hacer el usuario y cuando es el CPU, el timer actúa y llama a las funciones de NASM correspondientes para registrar el movimiento.

Hay que recordar que la matriz principal es gestionada en ensamblador, cada movimiento se manda a ensamblador en forma de offset de manera que la función verifica si esa casilla donde el usuario quiere jugar esta disponible y devuelve un resultado para continuar el proceso, dependiendo del resultado se decide que presentar en la interfaz gráfica.

```c
#ifndef ASM_UTILS_H
#define ASM_UTILS_H
/**
  Define the memory_managment function in asm
  - the first parameter is the offset of the button clicked
  - the second parameter is the variable to say who play
*/
extern int search_position(int offset);
extern void memory_setup();
extern long main_matriz[91];
#endif
```

Figure 17: Header de las funciones en NASM.

```c
#include <asm_utils.h>
#include <stdio.h>

long record_movement(long offset) {
    long result;

    printf("\nPre move:\n");
    for (int i = 0; i < 91; i++) {
        printf("%ld ", main_matriz[i]);
    }

    result = search_position(offset);

    if (result == 0) {
        printf("Invalid move: Position already occupied.\n");
        return -1;
    }

    printf("\nPost move:\n");
    for (int i = 0; i < 91; i++) {
        printf("%ld ", main_matriz[i]);
    }

    printf("Player %ld played at position %ld\n", result, offset);

    return result;
}
```

Figure 18: Funcion para registrar y validar un movimiento en NASM.

```
section .text
global search_position

extern main_matriz

search_position:
    push rbp                    ; Save the base pointer
    mov rbp, rsp                ; Set the stack pointer as the base pointer
    sub rsp, 16                 ; Reserve space for local variables

    push rbx
    push rdi
    push rsi
    push rdx

    mov rsi, [rbp+24]           ; Load the offset into rsi (0-indexed)
    lea rcx, [rel main_matriz]  ; Get the base address of the matrix

    ; Check if the offset is valid (should not exceed the matrix size)
    cmp rsi, 81                 ; Compare the offset with the size of the matrix
    jge invalid_move            ; If the offset is greater or equal to 81, it's invalid
```

Figure 19: Definicion de la funcion en NASM como global.

4 Diseño de la interfaz

La mayor parte de los elementos de la interfaz fueron acomodados usando Glade el cual es un diseñador que permite acomodar de manera grafica los elementos o widgets de gtk, sin embargo hubieron cosas las cuales se hicieron manualmente como lo anterior en la figura 13 donde se puede observar que se hacen los botones de manera manual para tenerlos en una matriz que apunten a cada uno de ellos, no obstante es importante tener en cuenta que las ventanas, los contenedores, algunos botones y todas la labels son echas y acomodadas desde glade. [3] [2]

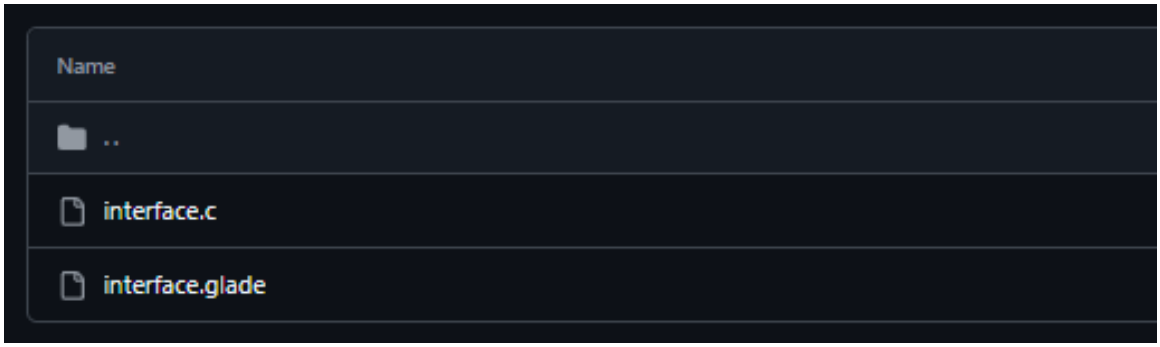

Figure 20: Archivo de configuracion de glade.

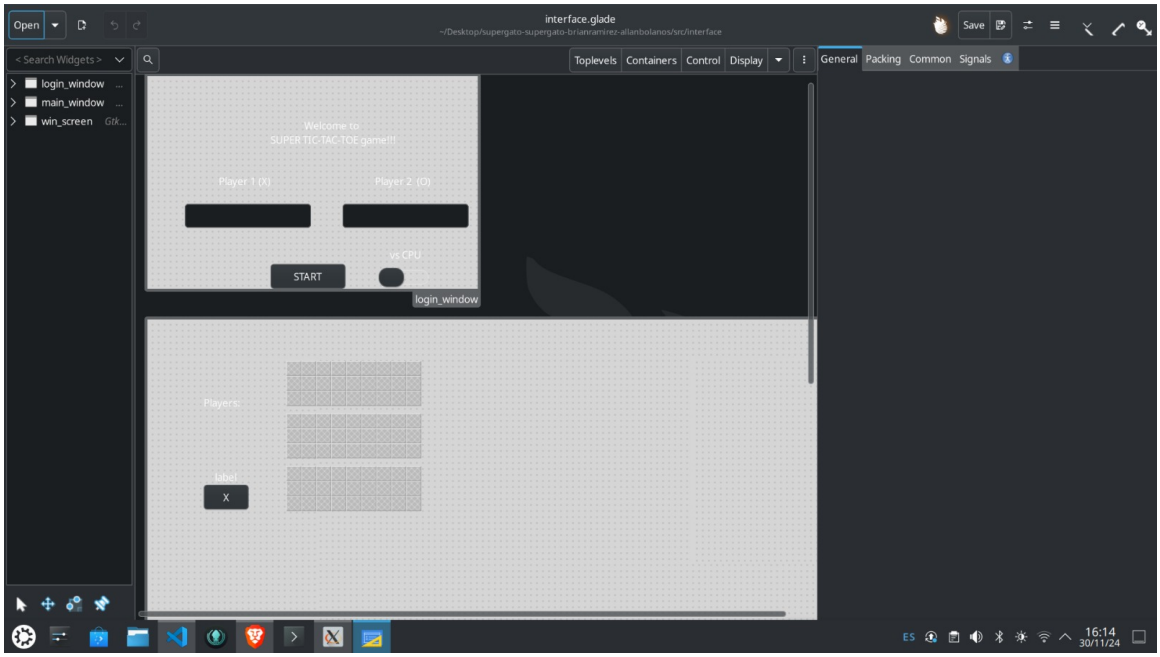

Figure 21: Desarrollo desde Glade.

[3]

5 NASM CODE:

5.1 Sección de Texto

La sección de texto (`section .text`) contiene el código ejecutable. La función expuesta es `search_position`.

```
section .text
global search_position
extern main_matriz
```

5.2 Función Principal: `search_position`

- **Descripción**: Verifica si un movimiento en la matriz es válido, actualiza la matriz con el movimiento del jugador y determina si hay un ganador.
- **Parámetros**:
  - El primer parámetro se pasa a través de la pila y representa la posición en la matriz donde se desea realizar el movimiento (offset 0-indexado).
- **Retorno**:
  - Devuelve un valor en `rax` que indica el resultado del movimiento:
    - `0`: Movimiento inválido.
    - `1`: Movimiento válido realizado por el jugador 1.
    - `2`: Movimiento válido realizado por el jugador 2.
    - `3`: Victoria para el jugador 1.
    - `4`: Victoria para el jugador 2.
    - `5`: Victoria general (cualquier jugador).

5.3 Detalles de Implementación

5.3.1 1. Configuración Inicial

Se guardan los registros utilizados y se reserva espacio en la pila para variables locales. Se carga el offset de la posición en `rsi` y se obtiene la dirección base de la matriz `main_matriz`.

```
search_position:
    push rbp ; Guardar el puntero base
    mov rbp, rsp ; Establecer el puntero de pila como el puntero base
    sub rsp, 16 ; Reservar espacio para variables locales

    push rbx
    push rdi
    push rsi
    push rdx

    mov rsi, [rbp+24] ; Cargar el offset en rsi (0-indexado)
    lea rcx, [rel main_matriz] ; Obtener la direccin base de la matriz
```

5.3.2 2. Verificación de Movimiento

Se multiplica el offset por 8 para calcular la posición correcta en bytes. Se carga el valor de la celda correspondiente en `rax`. Si el valor es 0, la celda está libre; si no, se salta a `invalid_move`.

```
    shl rsi, 3 ; Multiplicar rsi por 8 (equivalente a rsi * 8)
    mov rax, [rcx+rsi] ; Cargar el valor de la celda (64 bits)
    test rax, rax ; Si el valor es 0, la celda est vaca
    jnz invalid_move ; Si la celda est ocupada, saltar a invalid_move
```

5.3.3 3. Actualización de la Matriz

Se verifica quién es el jugador actual (1 o 2) y se actualiza la celda correspondiente en la matriz con el valor del jugador. Se cambia el turno del jugador actual.

```
    lea rdx, [rel main_matriz]
    mov rbx, [rdx+90*8] ; Cargar el valor del jugador actual en rbx

    cmp rbx, 1
    je player_1_move ; Si es jugador 1, procesar su movimiento

    cmp rbx, 2
    je player_2_move ; Si es jugador 2, procesar su movimiento
```

5.3.4 4. Comprobación de Ganador

Se realizan varias comprobaciones en filas, columnas y diagonales para determinar si hay un ganador. Si hay un ganador, se actualiza la matriz y se retorna el valor correspondiente.

```
check_win:
    lea rdx, [rel main_matriz] ; Direccin base de la matriz en rdx
    ; Comprobaciones para determinar el ganador
    ; (Cdigo omitido para brevedad)
```

5.3.5 5. Manejo de Errores

Si se intenta realizar un movimiento en una celda ocupada, se retorna `0`, indicando que el movimiento es inválido.

```
invalid_move:
    mov rax, 0 ; Resultado 0 indica movimiento invlido
    jmp exit
```

5.4 Etiquetas y Secciones Clave

- **Etiquetas de Jugador**:
  - `player_1_move`: Maneja el movimiento del jugador 1.
  - `player_2_move`: Maneja el movimiento del jugador 2.
  - `player_1_victory`: Maneja la victoria del jugador 1.
  - `player_2_victory`: Maneja la victoria del jugador 2.
- **Etiquetas de Comprobación de Ganador**:
  - `check_win`: Comprueba si hay un ganador después de un movimiento.
  - `check_win_row2`, `check_win_row3`: Comprobaciones específicas de filas.
  - `check_win_column`: Comprobaciones de columnas.
  - `check_win_diagonal`: Comprobaciones diagonales.

5.5 Ejemplo de Comprobación de Ganador

```
check_win:
    lea rdx, [rel main_matriz] ; Direccin base de la matriz en rdx
    mov r13, r9 ; Copia r9 (ndice) a r8
    imul r13, r13, 72 ; Calcul a el desplazamiento: r8 = r9 * 72
    add rdx, r13 ; Suma el desplazamiento a la direccin base

    mov r10, [rdx] ; Carga el primer elemento
    mov r11, [rdx+8] ; Carga el segundo elemento (8 bytes de distancia
       ↪  para 64 bits)
    mov r12, [rdx+16] ; Carga el tercer elemento (16 bytes de
       ↪ distancia)

    ; Comprobar si los tres valores son iguales (y no son 0)
    cmp r10, r11
    jne check_win_row2 ; Si r10 != r11, no hay victoria
    cmp r11, r12
    jne check_win_row2 ; Si r11 != r12, no hay victoria
    cmp r10, 0
    je check_win_row2 ; Si r10 es 0, no hay victoria
    ; Si todos son iguales, se ha encontrado un ganador
    mov rax, 3 ; Retornar victoria para el jugador 1
    ret
```

```
check_win_row2:
    ; Comprobaciones para la segunda fila
    ; (Cdigo omitido para brevedad)
    ret

\subsection{Salida}
La funcin finaliza limpiando la pila y restaurando los registros
    ↪ utilizados. El valor de retorno en \texttt{rax} indica el
    ↪ resultado del movimiento o el estado del juego.

\begin{lstlisting}[language=asm]
exit:
    pop rdx
    pop rsi
    pop rdi
    pop rbx
    mov rsp, rbp
    pop rbp
    ret
```

Bibliografía

[1] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1988.

[2] GTK Team, *GTK Documentation*, GNOME Project, 2024.

[3] GNOME Project, "Glade interface designer," 2024.

[4] S. P. Dandamudi, *Guide to Assembly Language Programming in Linux*, Springer, Ottawa, ON K1S5B6 Canada, 2005.