Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

2021/2022

Architecture des ordinateurs

Rapport 05

Travail pratique 5 : La tour de Hanoï en assembleur

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Branche	Architecture des ordinateurs
Etudiants	Barras Simon, Terreaux Nicolas
Groupe	B01
Classe	ISC-IL-2d
Professeur	Supcik Jacques
GitLab	https://gitlab.forge.hefr.ch/ado/2021-2022/classe-supcik/groupe-B-01/tp05
Version/Date	V1 du 25.01.2022

1 Introduction

Ce TP consiste à réaliser le jeu de la tour d'Hanoï sur l'écran LCD. Dans un premier temps, le jeu est fait en C et lorsque tout est fonctionnel, en assembleur. Les anciens TP sont très importants pour la réalisation de ce projet car il faut récupérer une partie du code pour le Rotary et pour le 7-Segment.

2 Résumés

Ce chapitre contient les différents points non-acquis, à exercer et parfaitement acquis durant le projet.

2.1 Non acquis

Nous ne pensons pas qu'il y ait de points non-acquis.

2.2 A exercer

Assembleur :

Nous avons bien compris le fonctionnement de l'assembleur mais nous ne nous sentons pas encore très à l'aise. Comme pour tous les autres langages, la pratique nous aiderait à être plus efficaces.

Architecture :

Nous sommes contents car nous comprenons de mieux en mieux l'architecture d'un programme en C/C++ ainsi que de savoir quand et où aller chercher les différentes méthodes et attributs.

2.3 Parfaitement acquis

Tests unitaires :

Nous avons désormais bien compris les tests unitaires et nous nous sentons à l'aise pour les faire. De plus, les tests avec les CI/CD fonctionnent correctement (nous avons pu le voir durant le TP sur les cibles à dispositions).

3 Points importants

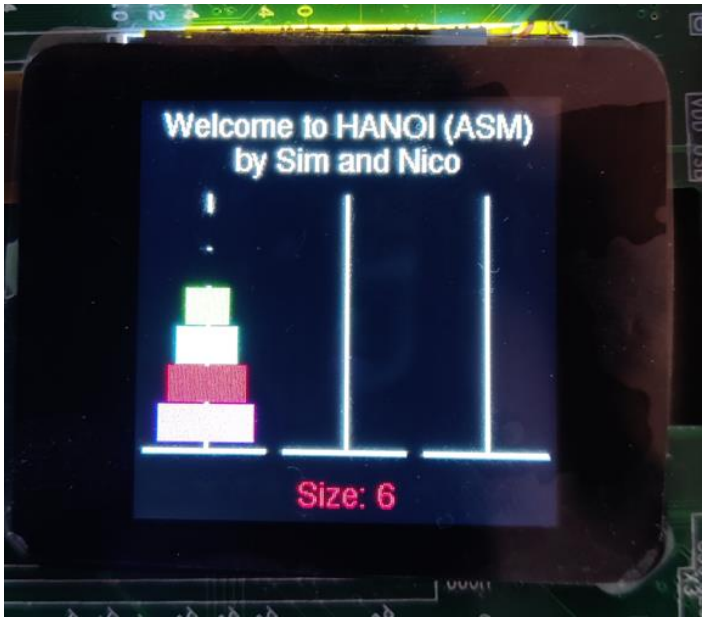
Ce chapitre liste tous les points qui nous sembles le plus important abordés durant ce TP.

3.1 Assembleur

La partie en C est plus ou moins trivial contrairement à la partie en assembleur. En effet, nous n'avons jamais codé en assembleur dans un vrai projet. La difficulté est de "changer" sa façon de coder et de prendre les opérations une à une.

Problème :

Durant l'implémentation, les deux disques du haut ne veulent pas s'afficher. Nous avons passé beaucoup de temps sur le problème en faisant du debug et nous avons remarqué que les deux derniers disques nous donnent la couleur 0 en hexadécimal ce qui les affiche en noir comme on peut le voir sur l'image suivante :



En faisant du debug, nous avons remarqué qu'à la ligne 266, le programme mettait 0 dans r3 pour les deux disques du haut (Index 0 et 1).

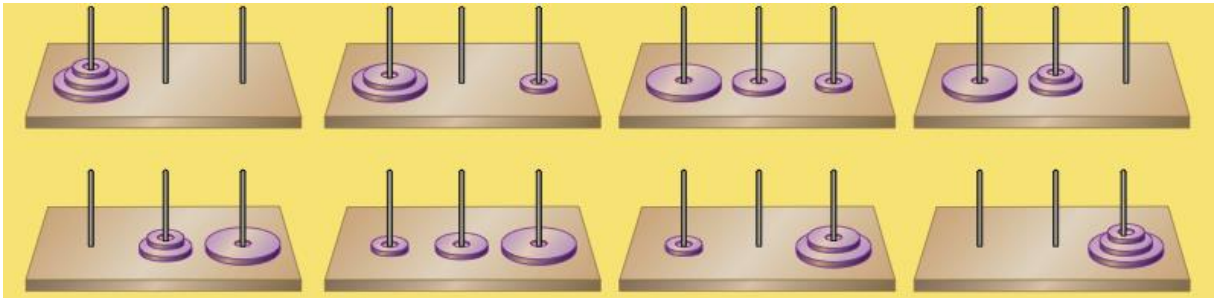
```
260
261     // canvas_rectangle(x, height, width, disk_height_, disk_color[disk]);
262     subs    sp, #1 * 4
263     ldr     r5, =disk_color
264     mov     r3, #4
265     mul     r6, r4, r3
266     ldr     r3, [r5, r6]
267     str     r3, [sp]
268     ldr     r3, =disk_height
269     ldrb    r3, [r3]
270
271     bl      canvas_rectangle
```

Solution :

Suite à une discussion avec le professeur, nous avons résolu le problème en alignant correctement la stack et en faisant les modifications nécessaires dans les instructions (ldr).

3.2 Algorithme de Hanoï

L'algorithme de Hanoï nous était déjà familier car nous l'avions déjà réalisé dans le cours d'Algorithmique 1. C'est un algorithme récursif nous permettant de déplacer des tours de disques d'un point A à un point B.



https://accromath.uqam.ca/2016/02/les-tours-de-hanoi-et-la-base-trois/

3.3 CI/CD

Durant ce TP, la particularité du CI/CD est qu'il est configuré pour effectuer des tests unitaires directement sur une cible mise à disposition par les professeurs.

3.4 Tests unitaires

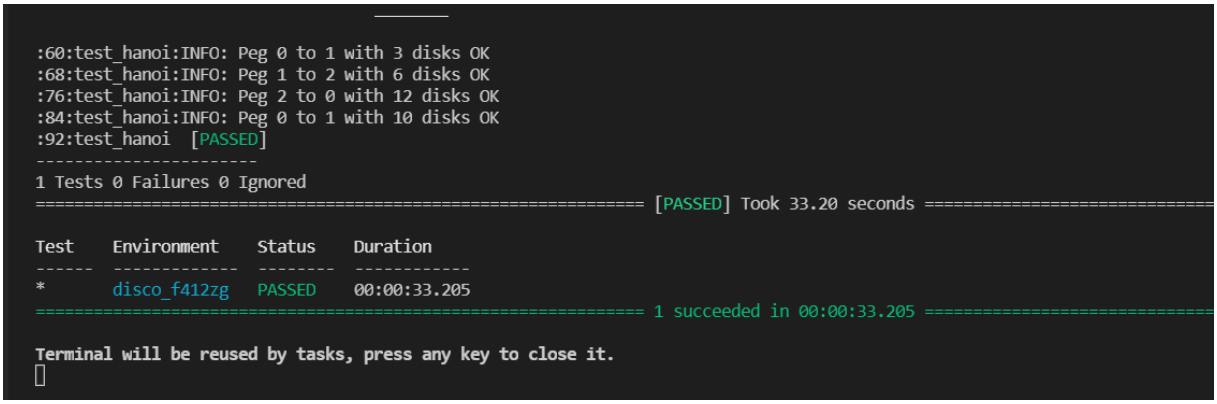
Le but de nos tests unitaires est de valider le nombre de mouvements nécessaires pour déplacer une tour de n disques.

Nous avons réalisé différentes variantes :

```
void test_hanoi(void)
{
    tower_of_hanoi_init(0, 3);
    wait_2_sec();
    int moves = tower_of_hanoi_move(0, 1, 2, 3);
    display_moves(moves);
    TEST_ASSERT_EQUAL(7, moves);
    TEST_MESSAGE("Peg 0 to 1 with 3 disks OK");

    wait_2_sec();
    tower_of_hanoi_init(1, 6);
    wait_2_sec();
    moves = tower_of_hanoi_move(1, 2, 0, 6);
    display_moves(moves);
    TEST_ASSERT_EQUAL(63, moves);
    TEST_MESSAGE("Peg 1 to 2 with 6 disks OK");

    wait_2_sec();
    tower_of_hanoi_init(2, 12);
    wait_2_sec();
    moves = tower_of_hanoi_move(2, 0, 1, 12);
    display_moves(moves);
    TEST_ASSERT_EQUAL(4095, moves);
```

```
    TEST_MESSAGE("Peg 2 to 0 with 12 disks OK");

    wait_2_sec();
    tower_of_hanoi_init(0, 10);
    wait_2_sec();
    moves = tower_of_hanoi_move(0, 1, 2, 10);
    display_moves(moves);
    TEST_ASSERT_EQUAL(1023, moves);
    TEST_MESSAGE("Peg 0 to 1 with 10 disks OK");
}
```

Et voici le résultat :



4 Conclusion

Nous avons bien aimé faire ce projet. Le jeu d'Hanoï nous était déjà familier et nous avons donc tout de suite compris comment il fallait aborder le TP. De plus, nous avions une base solide grâce aux anciens TP que nous avions bien réalisé ce qui nous a fait gagner du temps lors de l'implémentation du rotary et du 7-segment.

La partie assembleur était très intéressante car c'est la première fois que nous voyons concrètement un résultat de notre code. Et c'est assez satisfaisant lorsque le LCD affiche ce que nous voulons. Nous avons cependant eu le problème cité dans le chapitre assembleur que nous avons réussi à résoudre grâce à l'aide du professeur. Nous sommes très contents d'avoir un projet totalement fonctionnel avec la modification de la taille de la tour.

Le fait d'avoir configurer un CI/CD qui effectue les tests unitaires sur une cible distante rajoutait une motivation supplémentaire.

Nous avons passé chacun environ **14** heures en dehors des cours pour terminer ce projet.

5 Sources

tower_asm.S :

```
// Copyright 2022 Haute école d'ingénierie et d'architecture de Fribourg
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
```

```
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/*****************************************************************************
 * @file tower_asm.S
 * @author Barras Simon <simon.barras@edu.hefr.ch>
 * @author Terreaux Nicolas <nicolas.terreaux@edu.hefr.ch>
 *
 * @brief Tower resolution with assembly
 *
 * @date 2022-01-25
 * @version 0.1.0
 *****************************************************************************/

#ifdef VARIANT_ASM

//! 2 - Includes
#include "constants.h"
#include "color.h"
#include "canvas.hpp"

// 3 Symboles (textual expressions)

// 4 Declaration of constants
    .section .rodata
    .align 2
    WELCOME_TEXT: .asciz "Welcome to HANOI (ASM)"
    BY_TEXT: .asciz "by Sim and Nico"

// Ex: MY_CONSTANT:    .long 10

// 5 Declaration of variables with non-zero initial value
    .section .data
    .align 2
    nb_disks:           .byte 10

// 6 Declaration of variables with zero initial value
    .section .bss
    .align 2
    disk_color:         .space 4 * NB_OF_DISKS_MAX
    .align 2
    pegs_occupation:    .space 4 * NUMBER_OF_PEGS
    .align 2
    disk_height:        .space 4
```

```
// 7 Function implementation
    .section .text
    .thumb
    .syntax unified
    .align 2

    .global tower_of_hanoi_init
    .global draw_peg
    .global push_disk
    .global draw_disk
    .global clear_disk
    .global tower_of_hanoi_move
    .global move_disk


//*****************************************************************************
*
    .type tower_of_haoi_init, %function
// void tower_of_hanoi_init(int peg, int nb_disks)
tower_of_hanoi_init:
    push    {r4-r5, lr}

    // canvas_init();
    mov     r4, r0
    ldr     r3, =nb_disks
    strb    r1, [r3]
    bl      canvas_init
    // MakeRainbow(&disk_color[0], nb_disks);
    ldr     r1, =nb_disks
    ldrb    r1, [r1]
    ldr     r0, =disk_color
    bl      MakeRainbow

    // for (int i = 0; i < NUMBER_OF_PEGS; i++) {
    //     pegs_occupation_[i] = 0;
    //  }
    mov     r0, #0
    mov     r5, #0
    ldr     r1, =pegs_occupation
    b 2f
1:  mov     r2, #0
    strb    r2, [r1, r5]
    add     r0, #1
    add     r5, #4
2:  cmp     r0, #NUMBER_OF_PEGS
    blt     1b
```

```
    // disk_height_ = (PEG_HEIGHT - (nb_disks + 1) * DISK_MARGE - 10) /
nb_disks;
    ldr     r0, =PEG_HEIGHT - 1
    ldr     r3, =nb_disks
    ldrb    r3, [r3]
    sub     r0, r3
    ldr     r5, =DISK_MARGE
    mul     r0, r5
    sub     r0, #10
    sdiv    r2, r0, r3
    ldr     r3, =disk_height
    str     r2, [r3]

    // for (int i = NUMBER_OF_PEGS - 1; i >= 0; i--) {
    //     draw_peg(i);
    // }
    ldr     r5, =NUMBER_OF_PEGS - 1
    b 2f
1:  mov     r0, r5
    bl      draw_peg
    sub     r5, #1
2:  cmp     r5, #0
    bpl     1b

    // for (int i = nb_disks - 1; i >= 0; i--) {
    //     push_disk(peg, i);
    // }
    ldr     r5, =nb_disks
    ldrb    r5, [r5]
    sub     r5, #1
    b 2f
1:  mov     r0, r4
    mov     r1, r5
    bl      push_disk
    sub     r5, #1
2:  cmp     r5, #0
    bpl     1b

    // canvas_text_center(LCD_FONT_HEIGHT, "Welcome to HANOI (C)",
COLOR_WHITE);
    ldr     r0, =LCD_FONT_HEIGHT
    ldr     r1, =WELCOME_TEXT
    ldr     r2, =COLOR_WHITE
    bl      canvas_text_center

    // canvas_text_center(LCD_FONT_HEIGHT * 2, "By Sim and Nico",
COLOR_WHITE);
    ldr     r0, =LCD_FONT_HEIGHT * 2
    ldr     r1, =BY_TEXT
```

```
    ldr     r2, =COLOR_WHITE
    bl      canvas_text_center

    // display_size(nb_disks_);
    ldr     r0, =nb_disks
    ldrb    r0, [r0]
    bl      display_size

    pop     {r4-r5, pc}
    .size tower_of_hanoi_init, .-tower_of_hanoi_init


//*****************************************************************************
*
    .type draw_peg, %function
draw_peg:
    push    {r4-r5, lr}
    movs    r4, r0
    // Vertical bar
    // find x
    // (PEG_MARGE / 2 + (peg)*PEG_MARGE)
    subs    sp , #2 * 4

    ldr     r1, =COLOR_WHITE
    mov     r5, #0
    ldr     r1, [r1, r5, lsl #2]
    str     r1, [sp, #0]

    ldr     r0, =PEG_MARGE/2
    movs    r2, #PEG_MARGE
    mul     r1, r4, r2
    adds    r0, r0, r1
    movs    r1, #TOP
    movs    r2, #PEG_BORDER
    movs    r3, #PEG_HEIGHT
    movs    r5, r0
    bl      canvas_rectangle

    // Horizontal bar
    ldr     r1, =PEG_WIDTH/2
    subs    r0, r5, r1
    ldr     r1, =TOP + PEG_HEIGHT
    movs    r2, #PEG_WIDTH
    movs    r3, #PEG_BORDER
    bl      canvas_rectangle

    adds    sp, #2 * 4
    pop     {r4-r5, pc}
    .size draw_peg, .-draw_peg
```

```
//*****************************************************************************
*
    .type push_disk, %function
// static void push_disk(int peg, int disk)
push_disk:
    push    {r4-r6, lr}
    mov     r6, r0
    ldr     r4, =nb_disks
    ldrb    r4, [r4]
    mov     r2, r1
    // auto y = GET_DISK_HEIGHT(peg);
    // HEIGHT_MAX + (nb_disks_ - pegs_occupation_[peg] - 1) * (DISK_MARGE +
disk_height_)


    ldr     r3, =DISK_MARGE
    ldr     r4, =disk_height
    ldrb    r4, [r4]
    adds    r3, r4

    ldr     r4, =nb_disks
    ldrb    r4, [r4]
    ldr     r5, =pegs_occupation
    ldr     r5, [r5, r0, lsl #2]
    subs    r4, r5
    subs    r4, #1
    mul     r3, r4
    ldr     r4, =HEIGHT_MAX
    adds    r1, r3, r4

    // draw_disk(peg, y, disk);
    bl      draw_disk

    // pegs_occupation_[peg]++;
    ldr     r1, =pegs_occupation
    ldr     r3, [r1, r6, lsl #2]
    add     r3, #1
    str     r3, [r1, r6, lsl #2]

    pop     {r4-r6, pc}
    .size push_disk, .-push_disk


//*****************************************************************************
*
    .type draw_disk, %function
// static void draw_disk(int peg, int height, int disk)
```

```
draw_disk:
    push    {r4-r6, lr}
    mov     r4, r2
    // auto width = GET_DISK_WIDTH(disk);
    // DISK_SIZE_MIN + disk * DISK_SIZE_DIFF
    // #define DISK_SIZE_DIFF ((DISK_MAX_SIZE - DISK_SIZE_MIN) / nb_disks_)
    ldr     r5, =DISK_MAX_SIZE - DISK_SIZE_MIN
    ldr     r6, =nb_disks
    ldr     r6, [r6]
    sdiv    r5, r5, r6
    mul     r2, r5
    add     r2, DISK_SIZE_MIN

    // auto x     = GET_PEG_X(peg) - (width / 2);
    // PEG_MARGE / 2 + (peg)*PEG_MARGE
    ldr     r5, =PEG_MARGE
    lsr     r3, r5, #1
    mul     r0, r5
    add     r0, r3
    lsr     r3, r2, #1
    subs    r0, r3

    // canvas_rectangle(x, height, width, disk_height_, disk_color[disk]);
    subs    sp, #2 * 4
    ldr     r5, =disk_color
    ldr     r3, [r5, r4, lsl #2]
    str     r3, [sp]
    ldr     r3, =disk_height
    ldr     r3, [r3]

    bl      canvas_rectangle

    adds    sp, #2 * 4
    pop     {r4-r6, pc}
    .size draw_disk, .-draw_disk


//*****************************************************************************
*
    .type clear_disk, %function
//static void clear_disk(int peg, int height)
clear_disk:
    push    {r4-r8, lr}
    mov     r4, r0

//  pegs_occupation_[peg]--;
    ldr    r1, =pegs_occupation
    ldr    r3, [r1, r4, lsl #2]
    subs   r3, #1
```

```
    str     r3, [r1, r4, lsl #2]

//  auto x = GET_PEG_X(peg);
//  GET_PEG_X(peg) (PEG_MARGE / 2 + (peg)*PEG_MARGE)
    ldr     r1, =PEG_MARGE
    lsr     r0, r1, #1
    mul     r1, r4
    add     r0, r1

//  auto y = GET_DISK_HEIGHT(peg);
//  GET_DISK_HEIGHT(peg) (HEIGHT_MAX + (nb_disks_ - pegs_occupation_[peg] - 1)
* (DISK_MARGE + disk_height_))
    ldr     r1, =nb_disks
    ldr     r1, [r1]
    ldr     r3, =pegs_occupation
    ldr     r3, [r3, r4, lsl #2]
    subs    r1, r3
    subs    r1, #1
    ldr     r3, =DISK_MARGE
    ldr     r5, =disk_height
    ldrb    r5, [r5]
    adds    r5, r3
    mul     r1, r5
    ldr     r3, =HEIGHT_MAX
    adds    r1, r3

//  canvas_rectangle(x - (DISK_MAX_SIZE / 2), y, DISK_MAX_SIZE, disk_height_,
COLOR_BLACK);
    subs    sp, #1 * 4
    mov     r3, #COLOR_BLACK
    str     r3, [sp]
    ldr     r3, =disk_height
    ldrb    r3, [r3]
    ldr     r2, =DISK_MAX_SIZE
    mov     r5, r0
    lsr     r6, r2, #1
    sub     r0, r6
    mov     r6, r1
    mov     r7, r3
    bl      canvas_rectangle

//  canvas_rectangle(x, y, PEG_BORDER, disk_height_, COLOR_WHITE);
    mov     r3, #COLOR_WHITE
    str     r3, [sp]
    ldr     r2, =PEG_BORDER
    mov     r0, r5
    mov     r1, r6
    mov     r3, r7
    bl      canvas_rectangle
```

```
    adds   sp, #1 * 4
    pop    {r4-r8, pc}
    .size clear_disk, .-clear_disk




    .type move_disk, %function
//static void move_disk(int from, int to, int disk)
move_disk:
    push   {r4-r5, lr}
    mov    r4, r1
    mov    r5, r2

//  clear_disk(from, disk);
    mov     r1, r5
    bl     clear_disk

//  push_disk(to, disk);
    mov     r0, r4
    mov     r1, r5
    bl      push_disk

    pop     {r4-r5, pc}
    .size move_disk, .-move_disk



    .type tower_of_hanoi_move, %function
//int tower_of_hanoi_move(int from, int to, int by, int height)
tower_of_hanoi_move:
    push   {r4-r8, lr}
    mov    r4, r0
    mov    r5, r1
    mov    r6, r2
    mov    r7, r3
//  wait();
    bl     wait

//  int moves = 0;
    mov    r8, #0

//  if (height > 0) {
    cmp    r7, #0
    bls    1f

//      height--;
```

```
    sub     r7, #1

//      moves += tower_of_hanoi_move(from, by, to, height);
    mov     r0, r4
    mov     r1, r6
    mov     r2, r5
    mov     r3, r7
    bl      tower_of_hanoi_move
    add     r8, r0

//      move_disk(from, to, height);
    mov     r0, r4
    mov     r1, r5
    mov     r2, r7
    bl      move_disk

//      moves++;
    add     r8, #1

//      moves += tower_of_hanoi_move(by, to, from, height);
    mov     r0, r6
    mov     r1, r5
    mov     r2, r4
    mov     r3, r7
    bl      tower_of_hanoi_move
    add     r8, r0

//  return moves;
1:  mov     r0, r8

    pop   {r4-r8, pc}
    .size tower_of_hanoi_move, .-tower_of_hanoi_move


#endif
```

tower.c :

```
// Copyright 2022 Haute école d'ingénierie et d'architecture de Fribourg
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
```

```c
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/*****************************************************************************
 * @file tower.c
 * @author Barras Simon <simon.barras@edu.hefr.ch>
 * @author Terreaux Nicolas <nicolas.terreaux@edu.hefr.ch>
 *
 * @brief Tower of hanoi resolution with C
 *
 * @date 2021-12-14
 * @version 0.1.0
 *****************************************************************************/

#include "tower.h"

#include <DiscoConsole.h>
#include <stdio.h>

#include "canvas.hpp"
#include "color.h"
#include "constants.h"

#ifdef VARIANT_C
// CONSTANTS
#define GET_PEG_X(peg) (PEG_MARGE / 2 + (peg)*PEG_MARGE)
#define GET_DISK_HEIGHT(peg) \
    (HEIGHT_MAX + (nb_disks_ - pegs_occupation_[peg] - 1) * (DISK_MARGE +
disk_height_))
#define GET_DISK_WIDTH(disk) (DISK_SIZE_MIN + disk * DISK_SIZE_DIFF)

static unsigned int disk_color[NB_OF_DISKS_MAX];
static int nb_disks_ = 6;
static int disk_height_;
static int pegs_occupation_[NUMBER_OF_PEGS];

void tower_of_hanoi_init(int peg, int nb_disks)
{
    canvas_init();
    MakeRainbow(&disk_color[0], nb_disks);
    for (int i = 0; i < NUMBER_OF_PEGS; i++) {
        pegs_occupation_[i] = 0;
    }
    nb_disks_    = nb_disks;
    disk_height_ = (PEG_HEIGHT - (nb_disks + 1) * DISK_MARGE - 10) / nb_disks;
    printf("Taille: %d\n", disk_height_);

    for (int i = NUMBER_OF_PEGS - 1; i >= 0; i--) {
```

```
        draw_peg(i);
    }

    for (int i = nb_disks - 1; i >= 0; i--) {
        push_disk(peg, i);
    }

    // Title
    canvas_text_center(LCD_FONT_HEIGHT, "Welcome to HANOI (C)", COLOR_WHITE);
    canvas_text_center(LCD_FONT_HEIGHT * 2, "By Sim and Nico", COLOR_WHITE);
    display_size(nb_disks_);
}

/**
 * @brief
 *
 */
static void clear_disk(int peg, int height)
{
    pegs_occupation_[peg]--;
    // put black rectangle
    auto x = GET_PEG_X(peg);
    auto y = GET_DISK_HEIGHT(peg);
    canvas_rectangle(x - (DISK_MAX_SIZE / 2), y, DISK_MAX_SIZE, disk_height_,
COLOR_BLACK);
    // redraw peg
    canvas_rectangle(x, y, PEG_BORDER, disk_height_, COLOR_WHITE);
}

/**
 * method to draw a "disk" on given "peg" at the specified position "height"
 */
static void draw_disk(int peg, int height, int disk)
{
    auto width = GET_DISK_WIDTH(disk);
    auto x     = GET_PEG_X(peg) - (width / 2);
    canvas_rectangle(x, height, width, disk_height_, disk_color[disk]);
}

/**
 * @brief
 *
 */
void push_disk(int peg, int disk)
{
    // x,y,w,h,color
    // auto y = BOTTOM_HEIGHT - (disk + 1) * (disk_height_ + DISK_MARGE);
    auto y = GET_DISK_HEIGHT(peg);
    draw_disk(peg, y, disk);
```

```
    pegs_occupation_[peg]++;
}

/**
 * method to move a "disk" out of specified peg "from" to another one "to"
 */
static void move_disk(int from, int to, int disk)
{
    clear_disk(from, disk);
    push_disk(to, disk);
}

/**
 * @brief
 *
 *
 *
 * @param peg
 * @return * method
 */
// void draw_peg(int peg);
void draw_peg(int peg)
{
    // x,y,w,h,color
    canvas_rectangle(GET_PEG_X(peg), TOP, PEG_BORDER, PEG_HEIGHT,
COLOR_WHITE);
    canvas_rectangle(
        GET_PEG_X(peg) - PEG_WIDTH / 2, TOP + PEG_HEIGHT, PEG_WIDTH,
PEG_BORDER, COLOR_WHITE);
}

/**
 * @brief Resolve the problem of hanoi's towers
 *
 * @param from starting peg
 * @param to target peg
 * @param by
 * @param height height of the disk to move
 * @return int nb of moves
 */
int tower_of_hanoi_move(int from, int to, int by, int height)
{
    wait();
    int moves = 0;
    if (height > 0) {
        height--;
        moves += tower_of_hanoi_move(from, by, to, height);
        move_disk(from, to, height);
        moves++;
```

```
        moves += tower_of_hanoi_move(by, to, from, height);
    }
    return moves;
}
#endif

void display_moves(int moves)
{
    canvas_rectangle(0, LCD_SCREEN_HEIGHT - MARGIN_BOTTOM - 20, SCREEN_WIDTH,
40, COLOR_BLACK);
    char text[50];
    strcpy(text, "Moves: ");
    char moves_str[5];
    itoa(moves, moves_str, 10);
    strcat(text, moves_str);
    printf("%s\n", text);
    canvas_text_center(LCD_SCREEN_HEIGHT - MARGIN_BOTTOM, text, COLOR_RED);
}

void display_size(int size)
{
    canvas_rectangle(0, LCD_SCREEN_HEIGHT - MARGIN_BOTTOM - 20, SCREEN_WIDTH,
40, COLOR_BLACK);
    char text[50];
    strcpy(text, "Size: ");
    char size_str[5];
    itoa(size, size_str, 10);
    strcat(text, size_str);
    printf("%s\n", text);
    canvas_text_center(LCD_SCREEN_HEIGHT - MARGIN_BOTTOM, text, COLOR_RED);
}

void wait()
{
    for (int i = 0; i < 1000; i++) {
        asm("nop");
    }
}
```