# B1 - Unix System Programming

B-PSU-100

# my_sokoban

A warehouse keeper game

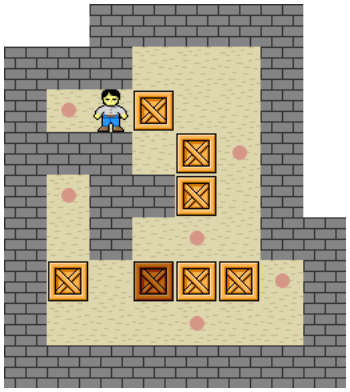
2.2

{EPITECH}

# my_sokoban

**binary name:** my_sokoban
**language:** C
**compilation:** via Makefile, including re, clean and fclean rules

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).



*Sokoban (warehouse keeper in Japanese) is a type of transport puzzle, in which the player pushes boxes or crates around in a warehouse, trying to get them to storage locations. The puzzle is usually implemented as a video game.*
*Sokoban was created in 1981 by Hiroyuki Imabayashi, and published in 1982 by Thinking Rabbit, a software house based in Takarazuka, Japan.*

*The game is played on a board of squares, where each square is a floor or a wall. Some floor squares contain boxes, and some floor squares are marked as storage locations. The player is confined to the board, and may move horizontally or vertically onto empty squares (never through walls or boxes). The player can also move into a box, which pushes it into the square beyond. Boxes may not be pushed into other boxes or walls, and they cannot be pulled. The puzzle is solved when all boxes are at storage locations.*

*Wikipedia*

Develop a copy of this game in terminal mode, using the ncurses library.
The map is not necessary square, it might have different shapes closed by a wall.

If all boxes are on storage locations, the player wins and the program must return 0. Otherwise, if none of the boxes can be moved anymore, he loses and the program must return 1. In any case, you do not need to print the last turn.

The game must be reset by pressing the space bar.
Redimensioning the terminal must be handled. As long as the terminal is too small to display the whole map, a centered message, asking the user to enlarge the terminal, must be displayed.

The game must be able to be played with the arrow keys (LEFT, RIGHT, UP and DOWN).

A valid map can only contains the characters SPACE, '\n', '#', 'X', 'O' and 'P'.
If an invalid map is supplied, the program must exit with an error.

Terminal

```
~/B-PSU-100> ./my_sokoban -h
USAGE
     ./my_sokoban map
DESCRIPTION
     map  file representing the warehouse map, containing ‘#’ for walls,
          ‘P’ for the player, ‘X’ for boxes and ‘O’ for storage locations.
```

Here is an example of map file:

Terminal

```
~/B-PSU-100> cat map
############
#        O #
#   P   ####
#          #
#######    #
#  O ##    #
#    ##    #
#          #
#       XX #
#       #  #
############
```

## AUTHORIZED FUNCTIONS

Every functions from the **ncurses** library are authorized.
In addition, the following system calls and functions are allowed:

- malloc, free, exit
- (f)open, (f)close, (f)read, (f)write
- getline, ioctl, usleep, sigaction, signal

- stat, lstat, fstat

## BONUS

You may add a lot of bonuses, such as:

- a full game menu, a game interface (with score, level etc.),
- time management,
- music, animations, tiling…
- a map editor, a map loader,
- AI,
- enhancement of the navigation using nice devices (gyroscope, joystick, leap motion, camera,…),
- a 2-player version, a network 2-player version,
- …

## UNIT TESTS

Testing a project which is using external libraries and system calls can be tedious, but it is possible to organize a project in such a way that external calls are isolated, and the core functionalities are easier to test. Also, architecturing your project to make it easily testable from the beginning usually leads to cleaner code. Here is an example:

```
#include <criterion/criterion.h>
#include "sokoban.h"

Test(sokoban, check_player_position)
{
    struct sokoban_map *map;
    struct position *pos;

    map = make_map_from_string("##########\n"\
                               "#        #\n"\
                               "#        #\n"\
                               "# PXO  ###\n"\
                               "#        #\n"\
                               "# OX     #\n"\
                               "##########\n");
    pos = get_player_position(map);

    cr_assert_neq(pos, NULL);
    cr_assert_eq(pos->x, 2);
    cr_assert_eq(pos->y, 3);

    free(pos);
    free_map(map);
}
```