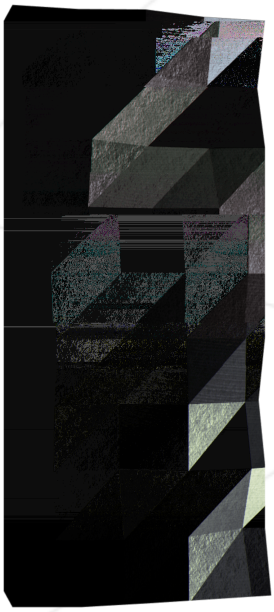

SoLong

And thanks for all the fish!

Summary:
This project is a very small 2D game.
Its purpose is to make you work with textures, sprites,
and some other very basic gameplay elements.

Version: 1

Contents

Chapter I

Foreword

Being a developer is a great thing for creating your own game.

But a good game needs some good assets. In order to create 2D games, you will have to search for tiles, tilesets, sprites, and sprite sheets.

Fortunately, some talented artists are willing to share their works on platforms like: itch.io

In any case, try to respect other people's work.

Chapter II

Goals

The goals of this project are similar to every other goal for this first year: being rigorous, level up in `C` programming, use basic algorithms, do some information research, and so forth.

`so long` is a graphic design project. It will help you improve your skills in the following areas: window management, event handling, colors, textures, and so forth.

Chapter III

Common Instructions

- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a `0` if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, use cc, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter IV

Mandatory part

Program name	so_long
Turn in files	Makefile, *.h, *.c, maps
Makefile	NAME, all, clean, fclean, re
Arguments	A map in format *.ber
External functs.	• open, close, read, write, printf, malloc, free, perror, strerror, exit • All the functions of the MiniLibX
Libft authorized	Yes
Description	You must create a small 2D game in which a dolphin escapes Earth after eating some fish. Instead of a dolphin, fish, and the Earth, you can use any hero, any collectible and any place you want.

Your project must comply with the following rules:

- You must use the `MiniLibX`. Either the version available on the school machines, or installing it using its sources.
- The management of your window must remain smooth (changing to another window, minimizing, and so forth).
- Although the given examples show a dolphin theme, you can create the world you want.

- The map has to be constructed with 3 components: walls, collectibles, and free space.
- The player's goal is to collect every collectible present on the map, then escape chosing the shortest possible route.
- At every move, the current number of movements must be displayed in the shell.
- The player should be able to move in these 4 directions: up, down, left, right.
- You have to use a 2D view (top-down or profile).
- The game doesn't have to be real time.
- The player should not be able to move into walls.
- Your program has to display the image in a window and must comply with the following rules:
  - The W, A, S, and D keys must be used to move the main character.
  - Pressing `ESC` must close the window and quit the program in a clean way.
  - Clicking on the red cross on the window's frame must close the window and quit the program in a clean way.
  - Using the `MiniLibX images` is strongly recommended.
- Your program has to take as first parameter a map description file ending with the `.ber` extension.
  - The map can be composed of only these 5 characters:
    0 for an empty space,
    1 for a wall,
    C for a collectible,
    E for a map exit,
    P for the player's starting position.

    Here is a simple valid map:

```
111111111111
100100000000C1
1000011111001
1P0011E000001
111111111111
```

  - The map must be closed/surrounded by walls. If it's not, the program must return an error.
  - Map must contain at least one exit, one collectible, and one starting position.
  - You don't have to check if there's a valid path in the map.

- The map must be rectangular.
- You must be able to parse any kind of map, as long as it respects the above rules.
- Another example of a minimal `.ber` map:

```
111111111111111111111111111111111
1E000000000000C00000C000000000001
101001010010000010100100000010101
101001001010101000100100000010101
1P0000000C00C000000000000000000C1
111111111111111111111111111111111
```

- If any misconfiguration of any kind is encountered in the file, the program must exit in a clean way, and return "Error\n" followed by an explicit error message of your choice.

Chapter V

Bonus part

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

You are allowed to use other functions to complete the bonus part as long as their use is **justified** during your evaluation. Be smart!

Bonus list:

- Make the player lose when they touch an enemy patrol.
- Add some sprite animation.
- Display the movement count directly on screen instead of using shell output.

You will be able to create better games later. Do not waste too much time on this project!

Chapter VI

Examples

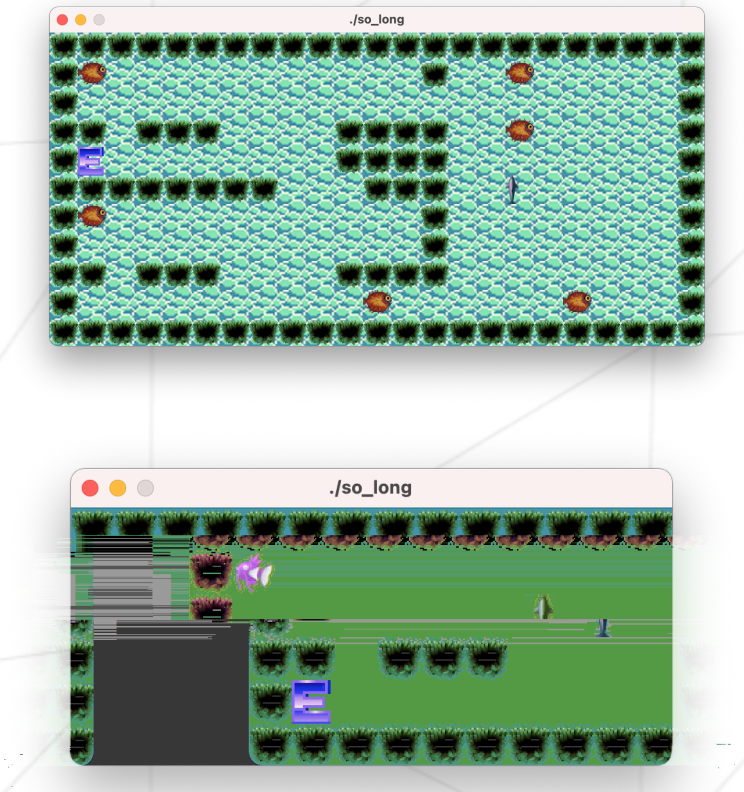

Figure VI.1: so_long examples showing terrible taste in graphic design (almost worth some bonus points)!