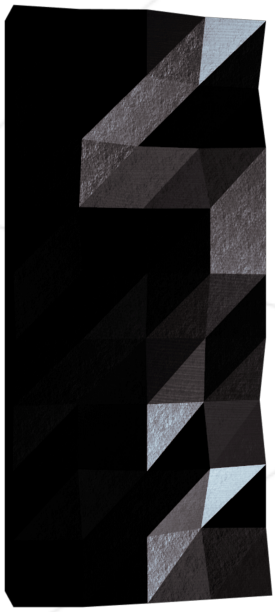


SoLong

And thanks for all the fish!

Resumen: Este proyecto es un pequeño juego en 2D. Está diseñado para hacerte trabajar con texturas y sprites y algunos otros elementos básicos de jugabilidad.

Versión: 2.3

Índice general

Capítulo I

Introducción

Ser desarrollador es genial si quieres crear tu propio juego.
Pero un buen juego necesita recursos gráficos buenos.
Para juegos 2D, debes buscar tiles, tilesets, sprites y sprite sheets.
Por suerte para ti, algunos artistas con talento están dispuestos a compartir su trabajo en plataformas como:
itch.io

Hagas lo que hagas, respeta el trabajo de otros.

Capítulo II

Objetivos

¡Llegó el momento de crear un proyecto de diseño gráfico básico!

`so_long` te ayudará a mejorar tus habilidades en estas áreas: ventanas, colores, eventos, texturas, etc.

Vas a usar la librería gráfica del campus `MiniLibX`. Esta librería se ha desarrollado internamente e incluye las herramientas básicas necesarias para abrir una ventana, crear imágenes y trabajar con eventos de teclado y ratón.

Los objetivos de este proyecto son similares a los de este primer año: rigor, uso de `C`, uso de algoritmos básicos, búsqueda de información, etc.

Capítulo III

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni tener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra` y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo IV

Parte obligatoria

Nombre de programa	`so_long`
Archivos a entregar	`Makefile, *.h, *.c, maps, textures`
Makefile	`all, clean, fclean, re, bonus`
Argumentos	`un mapa en formato *.ber`
Funciones autorizadas	• `open, close, read, write, malloc, free, perror, strerror, exit` • `Todas las funciones de la librería math (flag del compilador -lm, man 3 math` • `Todas las funciones de la miniLibX` • `ft_printf y cualquier función equivalente que tú hayas escrito`
Se permite usar libft	`Sí`
Descripción	`Debes crear un pequeño juego 2D donde un delfín escape del planeta Tierra después de comer pescado...En vez de un delfín, peces y la Tierra, puedes usar cualquier personaje, cualquier objeto y lugar que quieras.`

Tu proyecto debe cumplir con las siguientes normas:

- Debes usar la `miniLibX`. Ya sea la versión disponible en el sistema operativo, o su fuente. Si eliges trabajar con la fuente, deberás compilar siguiendo las mismas

normas que con tu `libft`, descritas en la parte de `Instrucciones generales`.

- Debes entregar un `Makefile` que compile con tus archivos fuente.
- Tu programa debe aceptar como primer argumento un archivo con la descripción del mapa de extensión `.ber`.

IV.1. Juego

- El objetivo del jugador es recolectar todos los objetos presentes en el mapa y salir eligiendo la ruta más corta posible.
- Las teclas W, A, S y D se utilizarán para mover al personaje principal.
- El jugador debe poder moverse en `4 direcciones`: subir, bajar, ir a la izquierda o ir a la derecha.
- El jugador no puede entrar dentro de las paredes.
- Tras cada movimiento, el `número real de movimientos`debe mostrarse en un terminal.
- Utilizarás una `perspectiva 2D` (vista de pájaro o lateral).
- El juego no necesita ser en tiempo real.
- Aunque los ejemplos dados se refieren a una temática de delfín, puedes crear el mundo que quieras.

`Si lo prefieres, puedes utilizar ZQSD o las teclas de dirección para mover el personaje principal.`

IV.2. Gestión de gráficos

- El programa mostrará la imagen en una ventana.
- La gestión de tu ventana debe ser limpia (cambiar de ventana, minimizar, etc)
- Pulsar la tecla `ESC` debe cerrar la ventana y cerrar el programa limpiamente.
- Hacer clic en la cruz roja de la ventana debe cerrar la ventana y terminar el programa limpiamente.
- El uso de `images` de la `miniLibX` es obligatorio

IV.3. Mapa

- El mapa estará construido de 3 componentes: `paredes`, `objetos` y `espacio abierto`.
- El mapa estará compuesto de solo 5 caracteres: **0** para un espacio vacío, **1** para un muro, **C** para un coleccionable, **E** para salir del mapa y **P** para la posición inicial del jugador.
  Este es un ejemplo simple de un mapa válido:

```
1111111111111
10010000000C1
1000011111001
1P0011E000001
1111111111111
```

- El mapa debe tener `una salida`, al menos `un objeto` y `una posición inicial`.

`Si el mapa contiene caractéres duplicados (salida o posición inicial), deberás mostrar un mensaje de error.`

- El mapa debe ser rectangular.
- El mapa deberá estar cerrado/rodeado de muros, en caso contrario el programa deberá devolver un error.
- Tienes que comprobar si hay un camino válido en el mapa.
- Debes poder procesar cualquier tipo de mapa, siempre y cuando respete las anteriores normas.
- Otro ejemplo minimalista de un mapa **.ber**:

```
1111111111111111111111111111111111
1E0000000000000C00000C000000000001
1010010100100000101001000000010101
1010010010101010001001000000010101
1P0000000C00C0000000000000000000C1
1111111111111111111111111111111111
```

- En caso de fallos de configuración de cualquier tipo encontrados en el archivo, el programa debe terminar correctamente y devolver “Error\n” seguido de un mensaje explícito de tu elección.

Capítulo V

Parte extra

Normalmente te animaríamos a que desarrollaras algunas funcionalidades extras originales tuyas. Sin embargo, habrá proyectos gráficos mucho más interesantes más adelante. ¡Te están esperando! ¡No pierdas demasiado tiempo en este encargo!
Se permite el uso de otras funciones para completar la parte extra,
siempre y cuando su uso `se justifique` durante la evaluación. Sé inteligente.

Conseguirás puntos extra si:

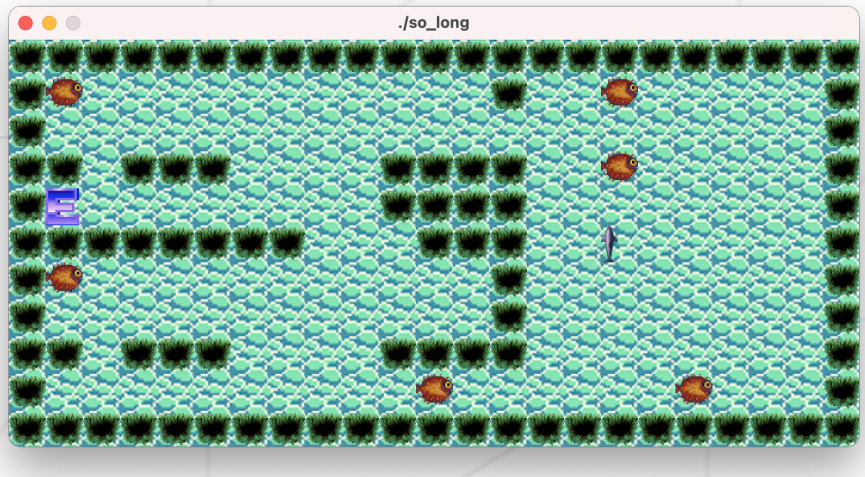
- Haces que el jugador pierda cuando toque una patrulla de enemigos
- Añades algunas animaciones de sprites.
- Muestras el contador de movimiento directamente en la pantalla en lugar de en el terminal.

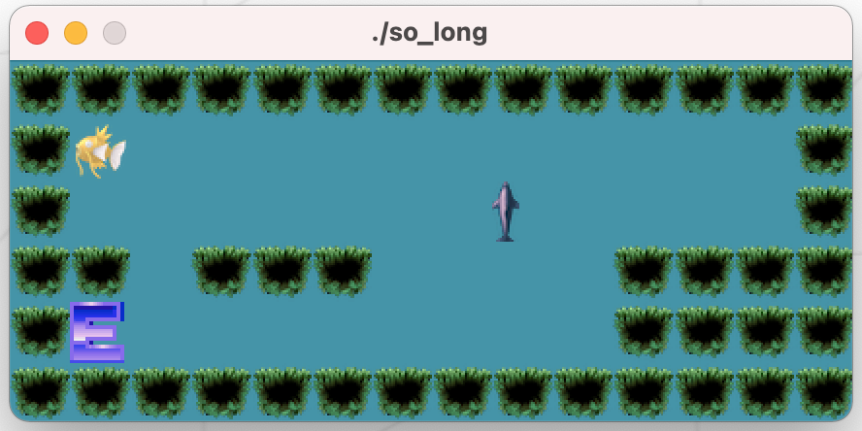
`Puedes añadir tantos archivos o carpetas como necesiten tus bonus.`

`La parte bonus será evaluada si y sólo si la parte obligatoria está PERFECTA. Perfecta quiere decir que toda la parte obligatoria funciona correctamente y la gestión de errores es impecable. Si no has superado todos los requisitos de la parte obligatoria, el bonus no será evaluado en absoluto.`

Capítulo VI

Ejemplos





Algunos ejemplos de `so_long` con muy mal gusto en diseño gráfico.
(Pero casi valen algunos puntos de bonus!)

Capítulo VII

Entrega y evaluación

Entrega tu trabajo en tu repositorio `Git` como de costumbre. Solo el trabajo de tu repositorio será evaluado. Comprueba dos veces si es necesario que los nombres de tus archivos sean los correctos.

Como estos ejercicios no se verifican con un programa, puedes organizar tus archivos como quieras, siempre que entregues los archivos obligatorios y cumplas los requisitos.