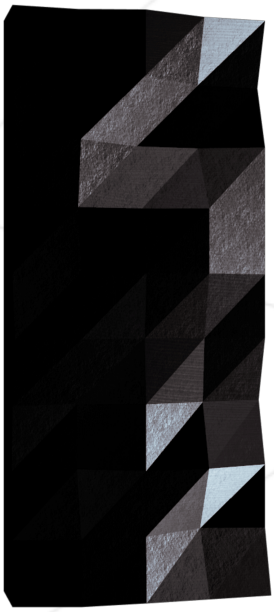


SoLong

Et merci pour le poisson !

Résumé:
Ce projet est un jeu 2D simple conçu pour vous faire utiliser des textures, des sprites et quelques éléments basiques de gameplay.

Version: 1

Table des matières

Chapitre I

Introduction

Être dev, c'est super pour créer son propre jeu. tiles carrelage structure de déplacement de mon perso

sprites = personnages

Cependant, un bon jeu nécessite de bonnes ressources. Afin de créer des jeux 2D, vous devrez rechercher des *tiles*, *tilesets*, des *sprites* et des *sprite sheets*.

Fort heureusement, des artistes de talent partagent leur travail sur des plateformes telles que :

itch.io

Bien entendu, veuillez respecter le travail d'autrui.

Chapitre II

Objectifs

Les objectifs de ce projet sont similaires à ceux de votre première année : faire preuve de rigueur, vous améliorer en programmation C, utiliser des algorithmes basiques, chercher des informations en autonomie, etc. ...

`so long` est un projet graphique. Il vous donnera des bases dans les compétences suivantes : gestion de fenêtre, gestion des événements, des couleurs, des textures, etc. ...

Chapitre III

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier différent : `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre IV

Partie obligatoire

Nom du programme	`so_long`
Fichiers de rendu	`Makefile, *.h, *.c, quelques cartes`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`Une carte au format *.ber`
Fonctions externes autorisées	• `open, close, read, write, printf, malloc, free, perror, strerror, exit` • `Toutes les fonctions de la MiniLibX`
Libft autorisée	`Oui`
Description	`Vous devez créer un jeu 2D basique dans lequel un dauphin s'échappe de la planète Terre après avoir mangé du poisson. Au lieu d'un dauphin, de poisson et de la Terre, vous pouvez utiliser le personnage, les items et le décor de votre choix.`

Votre projet doit respecter les règles suivantes :

- Vous **devez** utiliser la `MiniLibX`. Soit la version disponible sur les machines de l'école, soit en l'installant par les sources.
- La gestion de la fenêtre doit rester fluide (changer de fenêtre, la réduire, etc.).
- Bien que les exemples donnés montrent un thème dauphin, vous êtes libre de créer l'univers que vous voulez.

- La carte sera construite en utilisant 3 éléments : les murs, les items à ramasser, et l'espace vide.
- Le but du joueur est de ramasser tous les items présents sur la carte, puis de s'échapper en empruntant le chemin le plus court possible.
- À chaque mouvement, le compte total de mouvement doit être affiché dans le shell.
- Le joueur doit être capable de se déplacer dans ces 4 directions : haut, bas, gauche, droite.
- Vous devez utiliser une vue 2D (vue de haut ou de profil).
- Le jeu n'a pas à être en temps réel.
- Le joueur ne doit pas pouvoir se déplacer dans les murs.
- Votre programme doit afficher une image dans une fenêtre et respecter les règles suivantes :
  - Les touches W, A, S, et D doivent être utilisées afin de mouvoir le personnage principal.
  - Appuyer sur la touche `ESC` doit fermer la fenêtre et quitter le programme proprement.
  - Cliquer sur la croix en haut de la fenêtre doit fermer celle-ci et quitter le programme proprement.
  - L'usage des `images` de la `MiniLibX` est fortement recommandé.
- Votre programme doit prendre en paramètre un fichier de carte se terminant par l'extension `.ber`.
  - Votre carte peut être composée de ces 5 caractères :
    0 pour un emplacement vide,
    1 pour un mur,
    C pour un item à ramasser (*c* pour *collectible*),
    E pour une sortie,
    P pour la position de départ du personnage.

    Exemple de carte basique qui est correcte :

    ```
    1111111111111
    100100000000C1
    1000011111001
    1P0011E000001
    1111111111111
    ```

  - La carte doit être fermée en étant encadrée par des murs. Si ce n'est pas le cas, le programme retourne une erreur.
  - La carte doit contenir au moins une sortie, un item, et une position de départ.
  - Vous n'avez pas à vérifier s'il existe un chemin valide (qu'il est possible d'emprunter) dans la carte.

- La carte doit être de forme rectangulaire.
- Vous devez pouvoir parser tout type de carte du moment qu'elle respecte les règles énoncées ci-dessus.
- Un autre example de carte `.ber` basique :

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

- Si une erreur de configuration est détectée, le programme doit quitter proprement et retourner "Error\n" suivi d'un message d'erreur explicite de votre choix.

Chapitre V

Partie bonus

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Du moment que vous **justifiez** leur utilisation en évaluation, vous avez le droit d'utiliser des fonctions supplémentaires afin de faire la partie bonus. Soyez malins !

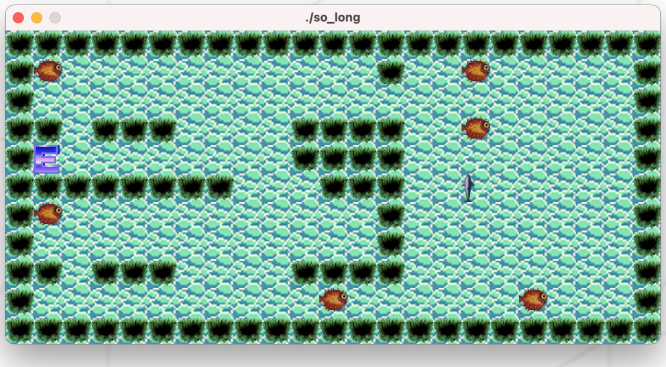
Liste de bonus :

- Le joueur peut perdre si son personnage est touché par une patrouille ennemie.
- Ajoutez des *sprite animations*.
- Le compte total de mouvement est directement affiché sur l'écran dans la fenêtre, plutôt que dans le shell.

Vous aurez l'occasion de créer des jeux plus poussés plus tard. Ne perdez donc pas trop de temps sur ce projet !

Chapitre VI

Exemples



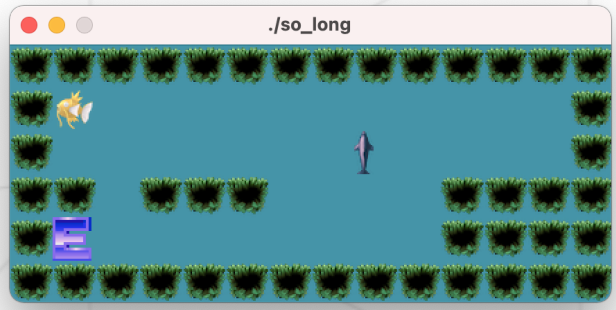


FIGURE VI.1 – Quelques exemples de `so_long` de très mauvais goût (ça pourrait presque compter comme bonus) !