Logo, company name

Description automatically generated

Cardinal chains

PROJET 1-CDEV

Documentation technique

**Auteurs :** Tommy BRISSET et Orlann FERREIRA

**Campus :** Tours

**Année scolaire :** 2022-2023

**Date :** 07-04-2023

Sommaire

* [Introduction](#introduction)
* [Organisation des fichiers](#Organisationdesfichiers)
* Choix techniques
* Conclusion
* Références

**Introduction**

Cardinal Chains est un jeu de puzzle avec un grille chiffrée ou il faut réaliser des chaines de couleurs avec des nombres logique croissants ou égaux.  
Dans chaque grille, il y a des cases contenant des « x » qui représente le point de départ d’une chaine. Il faut, à partir de ce point de départ, établir des chaines en reliant les cases via leurs valeurs , toujours avec des chiffres égaux ou croissant. Le but étant que l’ensemble des cases de la grille soit colorées.  
Cette version du jeu comporte 32 niveaux de difficultés croissantes.

Ce projet a pour objectif d’améliorer nos compétences en langage C a travers la réalisation d’un jeu de réflexion et de logique. Nous avons décider d’implémenter une interface graphique afin que l’utilisateur se sente plus à l’aise lorsqu’il joue a celui-ci. Cela nous a permis de découvrir les librairies graphique « Raylib » et « rayguy », qui permettent de faire des interfaces graphiques plus ou moins avancées en C. La création des différents niveaux du jeu n’étant pas générée automatiquement, nous avons donc du chercher un moyen de les enregistrez dans des fichiers textes et de les récupérer aux fur et à mesure de l’avancer de l’utilisateur dans l’expérience de jeu. Ce projet nous a aussi permit d’apprendre a mieux gérer la composition des différent fichier de code afin que la maintenance du code soit la plus simple et compréhensible pour le futur, sans oublier les commentaires du code insérer directement dans le code.

**Organisation des fichiers**

Dans notre rendu final, le dossier contenant tout les fichiers et autres documents est rangée selon les modalités suivantes :

* Un dossier **.vscode** contenant un fichier json qui est une configuration de fichiers associatifs pour l'éditeur de code Visual Studio Code. Cette configuration spécifie comment les fichiers avec des extensions spécifiques doivent être associés à des langages de programmation ou à des types de fichiers spécifiques dans l'éditeur de code.
* Un dossier **assets** qui contient les différentes images de notre programme, plus particulièrement une icone pour notre fenetre graphique « icon.png » et un « settings.pg » qui est l’image des paramètres du jeu.
* Un dossier **cmake-build-debug,** ce dossier contient les fichiers de construction temporaires générés lors du processus de génération. Lorsque l’on utilise CMake pour générer un projet C, on peut spécifier diverses options de configuration pour le projet, telles que la définition de variables, l'ajout de bibliothèques, la configuration de la langue de programmation, etc. Ces options de configuration sont stockées dans des fichiers de configuration générés par CMake. Ce dossier est utilisé pour compiler et exécuter nos programmes C en mode de débogage. En somme, ce dossier est un élément important de tout projet C généré avec CMake et est utilisé pour stocker les fichiers de construction temporaires nécessaires pour compiler et exécuter des programmes en mode de débogage dans un environnement de développement intégré.
* Un dossier **docs** contenant la documentation utilisateur et la documentation technique.

Un dossier **include** ayant comme contenu**,** les fichiers d'en-tête (en .h) définissant les interfaces pour les fonctions et les structures de données utilisées dans le code source, permettant au compilateur de vérifier la cohérence et la validité du code.  
Le dossier "include" est donc crucial pour le développement de notre programme C, car il garantit que toutes les dépendances du programme sont correctement incluses et référencées. En général, chaque fichier source (.c) est relié à un fichiers d'en-tête nécessaires à son fonctionnement qui indique au compilateur où trouver les définitions de fonctions et de variables.  
Cette organisation permet de faciliter la maintenance et l'extension du code, en limitant les dépendances entre les différents modules.  
le dossier "include" est donc un élément essentiel de notre projet C, car il contient les fichiers d'en-tête nécessaires à la compilation et à l'exécution du programme, assurant ainsi la validité et la cohérence du code.

* Un dossier **level** ou sont stockés tout les niveaux de notre jeu sous la forme de fichier « json », ces fichiers contiennent un liste qui représente notre grille de jeu. Les « 0 » représente une case de départ d’une chaine, les « -1 » représente une case vide que l’utilisateurs ne voit pas et tout les chiffres plus grand que zéro sont des valeurs permettant de faire des chaines

exemple de liste du niveau 2

* Un dossier **lib** qui contient un unique fichier "tinyfiledialogs.c". Ce fichier contient un ensemble de fonctions qui fournissent une interface simple pour l'affichage de boîtes de dialogue de fichiers sur différentes plates-formes, notamment Windows, macOS et Linux. Les fonctions permettent de demander à l'utilisateur de sélectionner des fichiers et des répertoires, d'ouvrir et de sauvegarder des fichiers, ainsi que d'afficher des messages d'erreur et d'avertissement. Ce code est distribué sous une licence libre qui permet son utilisation et sa modification pour tout usage.
* Un dossier **src** contenant tout les fichiers C qui permettent le fonctionnement du jeu. Celui ci est divisé en 5 fichiers :
* Un fichier « main.c », ce code est le programme principal de notre jeu. Il permet d’initialiser différentes variables tels que le statue courant du jeu, notre matrice, les différentes couleurs utilisée , et… On y retrouve la fonction qui charge les niveaux (initlevel), la fonction main qui fait tourner notre jeu et beaucoup de fonction qui nous permettent de récupérer les valeurs de nos variables.
* Un fichier « draw.c », qui contient les fonctions qui vont permette l’affichage graphique, on y retrouve, une fonction « DrawFrame » qui dessine la fenêtre principale du jeu en utilisant la bibliothèque graphique Raylib. Elle prend en paramètre un état "state" qui détermine quelle partie de l'interface doit être dessinée. Une fonction « DrawSettingsMenu » qui dessine une petite fenetre graphique contenant différent paramètre, une fonction « DrawCell » et « DrawBoard » qui dessinent le cellules (case) et le plateau de jeu.

* Un fichier gitignore, c’est un fichier texte qui permet de spécifier les fichiers ou les dossiers qui doivent être ignorés par Git lors de la prise en compte des modifications dans un projet. Git est un système de contrôle de version qui permet de suivre les modifications apportées à un projet de développement logiciel au fil du temps.

Un fichier **MakeList.txt**, c’est un fichier de configuration utilisé par le système de compilation CMake. CMake est un outil de compilation multiplateforme qui permet de générer des makefiles ou des projets pour différents environnements de développement. Il contient des informations sur les fichiers source, les bibliothèques externes nécessaires, les cibles de compilation, les options de compilation et les variables personnalisées. En utilisant CMake, notre projet peut est compatible avec différents systèmes d'exploitation et environnements.

* Un fichier **LICENSE.txt et README.md**, le premier permet de décrire les conditions d’utilisation, de modification, de distribution etc… Et le deuxième est une petite présentation du code, c’est-à-dire le principe du jeu ainsi qu’une petite notice d’installation du jeu sur son pc.

**RÉFÉRENCE**

Le code source ainsi que les différentes documentations du jeu sont disponibles sur GitHub via ce liens :   
  
code source : <https://github.com/Ferreira-Orlann/cardinal_chains_c_1st_year>  
license MIT : <https://fr.wikipedia.org/wiki/Licence_MIT>  
source tinifyledialogs : [tiny file dialogs (cross-platform C C++) download | SourceForge.net](https://sourceforge.net/projects/tinyfiledialogs/)