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

Cardinal Chains est un jeu de puzzle avec une grille chiffrée ou il faut réaliser des chaines de couleurs avec des nombres logiques croissants ou égaux.
Dans chaque grille, il y a des cases contenant des « x » qui représentent 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 croissants. Le but étant que l'ensemble des cases de la grille soit coloré.
Cette version du jeu comporte 32 niveaux de difficultés croissantes.

Ce projet a pour objectif d'améliorer nos compétences en langage C à travers la réalisation d'un jeu de réflexion et de logique. Nous avons décidé d'implémenter une interface graphique afin que l'utilisateur se sente plus à l'aise lorsqu'il joue. 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 dû chercher un moyen de les enregistrer dans des fichiers textes et de les récupérer au fur et à mesure de l'avancer de l'utilisateur dans l'expérience de jeu. Ce projet nous a aussi permis d'apprendre à mieux gérer la composition des différents fichiers de code afin que la maintenance de ce code soit la plus simple et compréhensible pour le futur, sans oublier les commentaires des différentes fonctions et attributs, insérer directement dans le code.

Organisation et contenu des fichiers

Dans notre rendu final, le dossier contenant tous les fichiers et autres documents est rangé 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 icône pour notre fenêtre graphique « icon.png » et un « settings.png » qui est l'image des paramètres du jeu.
- Un dossier **docs** contenant la documentation utilisateur et la documentation technique du projet.

- 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 tous les niveaux de notre jeu sous la forme de fichier « json », ces fichiers contiennent une 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 tous les chiffres plus grands 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 tous 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 statut courant du jeu, notre matrice, les différentes couleurs utilisées, et… On y retrouve la fonction qui charge les niveaux (initlevel), la fonction « main », qui fait tourner notre jeu et beaucoup d'autres fonctions 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, également, 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 fenêtre graphique contenant différents paramètres, une fonction « DrawCell » et « DrawBoard » qui dessinent le cellules (case) et le plateau de jeu. Une dernière fonction « DrawLevel » dessine le niveau actuel des joueurs.

  - Un fichier « save.c » qui contient les fonctions permettant de sauvegarder la progression de l'utilisateur et de reprendre plus tard. Plus précisément, la fonction « SaveLevel » enregistre les données d'un niveau du jeu dans un fichier avec l'extension ".save". Les données sauvegardées incluent : ID du niveau,l'indice de la dernière chaîne dans le niveau, les coordonnées de la dernière cellule, les chaînes et les types de cellules de la grille de jeu. La fonction « LoadLevel » charge les données d'un niveau de jeu à partir d'un fichier ".save" et les utilise pour restaurer l'état précédent du niveau. Les données chargées comprennent donc les mêmes que pour la fonction « SaveLevel ».

  - Un fichier « update.c », ou sont regroupées les différentes fonctions de mise à jour de notre jeu, on y retrouve une fonction « UpdateGame » qui met à jour le statut de notre jeu, une fonction « BoardUpdate » qui met à jour le plateau de jeu selon les actions du joueur. Nous avons aussi d'autres fonctions qui permettent de mettre à jour les différents boutons tels que « UpdateSaveButton » ou « UpdateLoadButton », une fonction « UpadteReiniLevel » qui remet la grille de jeu au départ (sans chaines de couleurs) si le joueur désire recommencer le niveau.

  - Un fichier « utils.c », avec des fonctions permettant de centrer nos textes dans la fenêtre graphique, « centerText » centre une chaine de caractères au coordonnées x et y, « CenterTextVec » centre à la position Vector2. Enfin , la fonction "DrawCenteredText" prend en entrée une chaîne de caractères, les coordonnées x et y, la taille de police et la

couleur de police. Elle récupère la taille de police par défaut, la modifie pour correspondre à la taille de police donnée. Elle calcule la position centrale à l'aide de la fonction « CenterText », calcule la taille du texte, dessine le texte et rétablit la taille de police par défaut à l'aide de « GuiSetStyle ».

- 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 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.

Choix technique

Ce projet devait être réalisé grâce aux langage C, nous avons utilisé Clion et visual studio code en tant qu'IDE. Afin de faciliter l'avancée du projet étant en groupe de 2 personnes, nous avons utilisé GitHub afin que chacun puisse avancer et mettre à disposition son code. Le code a été développée sur des PC tournant sous windows 10 et 11.
Nous avons structuré le code pour que celui soit le plus facile à comprendre et avons découpé les différentes parties du code dans des fichiers ayant une fonction bien définie (réf : Organisation des fichiers).

Structuration des niveaux : les différents niveaux implémentés sont enregistrés dans des fichiers json sous la forme suivantes (ex : [0,1,1,1]), une liste avec des valeurs différentes sachant que 0 représente un point de départ, -1 représente une case vide et toutes les autres valeurs (positives) permettent de faire des chaines. Avant d'insérer un niveau, il est important de vérifier que celui-ci est réalisable car nous ne procédons pas à une vérification dans le code en lui-même. Il est priorisé de faire des listes qui représente un plateau carré, même si celui-ci n'est pas carré à la fin car tous les niveaux ont été réalisé ainsi et cela permet de mieux comparée 2 niveaux différents.

Structuration des différents états du jeu : ceci nous permet de savoir ou l'utilisateur se situe dans notre jeu et d'afficher les bons graphismes selon le statut du jeu. Ils sont représentés par une structure, « GameState », qui est une énumération qui contient quatre constantes représentant les différents états du jeu : STATE_START, STATE_BOARD, STATE_WIN, STATE_LEVEL_FINISH.

```
typedef enum {
    STATE_START,
    STATE_BOARD,
    STATE_WIN,
    STATE_LEVEL_FINISH
} GameState;
```

Structuration des cases de notre plateau : La structure « Cell », est notre structure permettant de gérer les différentes cases de notre plateau de jeu Cardinal Chains. Elle contient plusieurs champs, une structure Rectangle représentant le rectangle associé à la cellule, une valeur de caractère, des coordonnées x et y, un identifiant de chaîne et un type de cellule représenté sous forme d'un octet.

```
// Size: 21 Bytes / 168 Bits
typedef struct {
    Rectangle rect; // 16 bytes (4 floats -> 4 * 4 Bytes) / 128 Bits
    unsigned char value;
    unsigned char x;
    unsigned char y;
    unsigned char chain;
    unsigned char type;
} Cell;
```

Structuration des types des cellules : « CellType », est une énumération qui définit les types de cellules possibles dans notre jeu Cardinal Chains. Les types de cellules sont représentés par des constantes qui ont chacune une valeur différente de puissance de deux, afin de permettre de les combiner entre elles à l'aide d'opérateurs binaires. Les types de cellules possibles sont : CELLTYPE_NONE, CELLTYPE_RIGHT, CELLTYPE_LEFT, CELLTYPE_TOP et CELLTYPE_BOTTOM.

```
typedef enum {
    CELLTYPE_NONE = (1 << 1),
    CELLTYPE_RIGHT = (1 << 2),
    CELLTYPE_LEFT = (1 << 3),
    CELLTYPE_TOP = (1 << 4),
    CELLTYPE_BOTTOM = (1 << 5),
} CellType;
```

Conclusion

Pour finir, ce projet nous a permis de développer nos connaissances dans le langage C, grâce à celui-ci nous avons pu mettre en pratique les notions de programmation vue durant nos cours sur un projet de jeu concret. L'implémentation de ce projet avec une interface graphique a aussi été très enrichissant au niveau de la programmation coté graphisme. La partie du projet portant sur la création des niveaux du jeu a elle aussi été très intéressante, nous avons dû réfléchir sur notre manière de faire des niveaux de difficultés croissantes tout en gardant un aspect de progression attractif pour l'utilisateur. Le projet final est donc fonctionnel et attractif pour l'utilisateur.

Bien que cette implémentation du jeu soit assez complète, certaines améliorations sont possibles et permettrait une expérience de jeu améliorée. Par exemple, une génération automatique des niveaux serait agréable pour l'utilisateur et cela permettrait une plus grande longévité du jeu. (Génération procédurale des niveaux par exemple)

Nous sommes satisfait du travail rendu et espérons que cette implémentation du jeu « Cardinal Chains » offre une expérience de jeu stimulante et attractive pour l'utilisateur.

RÉFÉRENCES

Le code source ainsi que les différentes documentations du jeu et les ressources externes sont disponibles via les liens ci-dessous :

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

raylib : https://www.raylib.com/

Clion : https://www.jetbrains.com/fr-fr/clion/

Visual studio code : https://code.visualstudio.com/download

Recoverable Signature

X Orlann Ferreira

Orlann Ferreira
Étudiant
Signed by: d00b22e8-6b95-4895-ab2e-797f4e8cc75f

Recoverable Signature

X Tommy Brisset

Tommy Brisset
Étudiant
Signed by: d00b22e8-6b95-4895-ab2e-797f4e8cc75f