Documentation C

Créé par : Clara Rabusseau

Date : 15 déc. 2024

Choix et du sujet

J’ai d’abord voulu réaliser une convertisseur de devises mais j’ai finalement choisi de créer une simulation du Jeu de la Vie de Conway en C. Ce projet met en œuvre l'automate cellulaire inventé par le mathématicien John Horton Conway en 1970. La simulation se déroule sur une grille où chaque cellule peut être vivante ou morte. À partir d'un état initial fixé par l'utilisateur, les cellules évoluent selon deux règles simples : la naissance d'une cellule vivante si trois cellules voisines sont vivantes, et la survie ou la mort d'une cellule vivante selon le nombre de ses voisines vivantes. Le jeu de la Vie est Turing complet c’est-à-dire qu’il est capable de simuler tout calcul qu'un ordinateur peut effectuer, à condition d'avoir une configuration initiale appropriée et un espace infini pour évoluer. Cela signifie qu'il peut représenter des algorithmes complexes, résoudre des problèmes logiques, et même simuler un ordinateur théorique, comme une machine de Turing.

Idées et démarches

Idées

Pour comprendre le fonctionnement du jeu, j’ai regardé des articles/sites sur le Jeu de la Vie de Conway mais aussi des vidéos Youtube comme celle du Youtubeur EGO :
🌐 Le Jeu de la Vie.

Elle montre bien la simplicité et en même temps la complexité et les perspectives d’évolutions de ce jeu. On peut obtenir différentes configurations : vaisseaux (Lightweight Spaceship) , Blinker, Structures fixes (bloc).

Démarches

I) Définition des structures et des paramètres par défaut :

J'ai organisé mon code pour le rendre le plus compréhensible possible. J'ai d'abord définit les structures et les paramètres par défaut. Il y a des paramètres globaux *comme le nombre de lignes, le nombre de colonnes, la taille des cellules, le nombre d'images par seconde, et le nombre d'itérations.*

Puis, j'ai créé une structure Grid pour représenter la grille du jeu. Cette structure contient le nombre de lignes et de colonnes, un tableau bidimensionnel dynamique pour représenter l'état des cellules (vivantes ou mortes).

II) Création et gestion de la grille :

J'ai implémenté plusieurs fonctions pour gérer la grille. La fonction create_grid alloue dynamiquement de la mémoire pour la grille et initialise toutes les cellules à 0 (mortes). La mémoire est allouée ligne par ligne pour permettre une gestion flexible. La fonction free_grid sert à libérer la mémoire allouée, évitant ainsi les fuites mémoire. Elles permettent de gérer efficacement la mémoire.

III) Initialisation de la grille avec des motifs :

J'ai ajouté la fonction initialize_grid, qui initialise la grille avec un motif spécifique. J'ai choisi de laisser l'option à l'utilisateur de choisir le motif qu'il souhaite dans les commandes d'exécution :

- *oscillator* : Un motif oscillant (3 cellules alignées horizontalement).
- *glider* : Un motif qui se déplace à travers la grille.
- *asterisk* : Un motif ponctuel.

Cette fonction est flexible grâce à l'utilisation de chaînes de caractères pour identifier les motifs. Si un motif inconnu est fourni, il y a un motif par défaut (oscillator) est utilisé.

IV) Gestion des règles du Jeu de la Vie :

La logique du Jeu de la Vie repose sur la gestion des voisins :

- count_neighbors : J'ai créé une fonction qui calcule le nombre de voisins vivants autour d'une cellule. Pour cela, j'utilise deux boucles imbriquées qui parcourent les 8 cellules environnantes, tout en évitant de compter la cellule elle-même.
- update_grid : J'ai également implémenté une fonction qui applique les règles du Jeu de la Vie : Une cellule vivante reste vivante si elle a 2 ou 3 voisins vivants. Une cellule morte devient vivante si elle a exactement 3 voisins vivants.

J'ai conçu cette partie de manière configurable en utilisant les tableaux survive et birth. Cela permet une personnalisation des règles du jeu en fonction des besoins ou des préférences de l'utilisateur.

V) Affichage avec Raylib :

Pour rendre la simulation visuelle, j'ai utilisé la bibliothèque Raylib. La fonction display_grid_raylib affiche la grille à chaque itération en utilisant des rectangles :

- Les cellules vivantes sont affichées en rose vif.
- Les cellules mortes sont représentées par des contours gris et des cases noires.

J'ai également défini la taille des cellules avec CELL_SIZE pour contrôler leur dimension dans la fenêtre.

VI) Simulation et gestion des entrées utilisateur :

J'ai ajouté la possibilité de configurer la simulation via des arguments comme –rows, –cols, –fps, et –pattern. Cela permet à l'utilisateur de personnaliser facilement l'expérience. Puis, j'ai créé et initialisé la grille en fonction des paramètres fournis. J'ai utilisé une boucle while pour exécuter la simulation tant que la fenêtre est ouverte. À chaque itération, la grille est mise à jour et affichée. J'ai également ajouté des contrôles comme la mise en pause ou la possibilité de quitter avec des touches spécifiques.

VII) Intégration de l'audio :

J'ai ajouté une fonctionnalité audio en utilisant Raylib :

- Initialisation de l'audio : J'ai utilisé InitAudioDevice() pour initialiser le périphérique audio.
- Chargement de la musique : J'ai utilisé LoadMusicStream pour jouer un fichier MP3 en arrière-plan.
- Lecture et mise à jour de la musique : J'ai utilisé PlayMusicStream et UpdateMusicStream.

J'ai essayé de modifier l'audio dans la boucle pour éviter les saccades mais ça n'a pas marché... J'ai voulu ajouter l'audio pour créer un rythme qui va avec la simulation du jeu de la Vie.

VIII) Nettoyage et libération des ressources :

À la fin du programme :

- J'ai libéré la mémoire allouée pour la grille avec free_grid.
- J'ai déchargé la musique avec UnloadMusicStream et fermé le périphérique audio avec CloseAudioDevice.
- J'ai également fermé la fenêtre avec CloseWindow.

Ces étapes garantissent que le programme ne laisse pas de ressources inutilisées après son exécution.

IX) Compilation et exécution :

Enfin, j’ai inclus les commandes nécessaires pour compiler et exécuter le programme :

- Compilation : J’ai utilisé gcc en liant correctement les bibliothèques Raylib.
- Exécution : J’ai ajouté différents paramètres pour tester divers motifs (–pattern glider, –pattern oscillator, etc.) et tailles de grilles.