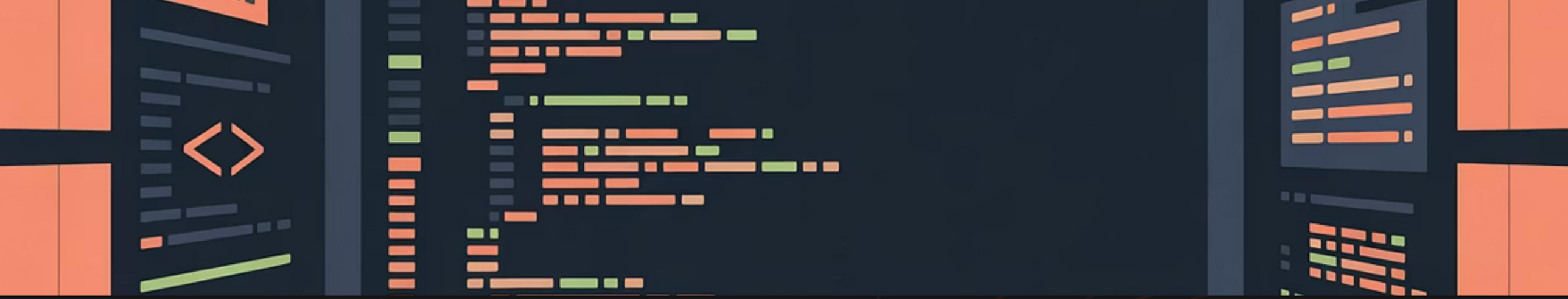
Snake en C – Console Windows

Un projet de jeu Snake classique développé en C pour console Windows, avec des mécaniques de gameplay innovantes et une gestion sophistiquée de la croissance du serpent.



Vue d'ensemble du projet

Langage et plateforme

Développé en C pour console Windows avec MSVC. Utilise les API Windows pour le contrôle du curseur et l'affichage en temps réel.

Mécaniques principales

Système de pommes multiples, croissance par séquence, gestion des vies avec réinitialisation, et temporisation lissée pour une expérience fluide.

Les quatre consignes fondamentales

3 pommes simultanées

Afficher trois pommes en même temps dans l'aire de jeu au lieu d'une seule.

Croissance par séquence

Le serpent grandit de +3 segments uniquement après avoir mangé les 3 pommes d'une séquence.

Réinitialisation à la mort

À chaque mort, la longueur du serpent revient à 5 segments et une nouvelle séquence de 3 pommes est générée.

Vitesse lissée

Utilisation d'une temporisation stable pour garantir une vitesse de jeu homogène sur toutes les machines.

Architecture des données

Le système repose sur des structures de coordonnées et des tableaux globaux pour gérer l'état du jeu en temps réel.

1

Structure coordinate

Contient les positions x, y et la direction. Utilisée pour la tête, les coudes, les pommes et le corps du serpent.

2

Tableau foods[3]

Stocke les trois pommes actives simultanément. Chaque élément peut être activé (coordonnées valides) ou désactivé (x=0, y=0).

3

Variables de comptage

apples (total de pommes mangées), applesStreak (compteur de séquence 0→3), length (longueur actuelle du serpent).

Génération des pommes multiples

Fonction SpawnApple()

Génère une pomme à un index donné dans la zone de jeu (entre x=11-70, y=11-30). Utilise rand() pour des positions aléatoires.

Appelée trois fois pour créer une séquence complète de pommes au démarrage et après chaque séquence terminée.

```
void SpawnApple(int idx){ foods[idx].x = rand() % 70; if (foods[idx].x <= 10)  foods[idx].x += 11; foods[idx].y = rand() % 30; if (foods[idx].y <= 10)  foods[idx].y += 11;}
```

Affichage des pommes actives

Dans la fonction Boarder(), les trois pommes sont affichées si elles sont actives (coordonnées non nulles). Chaque pomme est représentée par le caractère 'F'.

Le système vérifie foods[i].x != 0 avant d'afficher chaque pomme, permettant une gestion dynamique des pommes mangées.

Logique de croissance par séquence

01

Détection de collision

La fonction Food() vérifie si la tête du serpent touche une pomme active.

02

Incrémentation des compteurs

apples++ (score total) et applesStreak++ (progression dans la séquence).

03

Désactivation de la pomme

foods[i].x = 0 et foods[i].y = 0 pour retirer la pomme mangée de l'affichage.

04

Vérification de séquence

Si applesStreak == 3, le serpent grandit de +3 segments d'un coup.

05

Nouvelle séquence

applesStreak = 0 et génération de 3 nouvelles pommes via SpawnApple().

Code de la croissance

```
if (applesStreak == 3) {    for (int k = 0; k < 3 &&          length < (int)(sizeof(body)/
sizeof(body[0])) - 1; k++) {        length++; /* +3 segments */    }    applesStreak = 0;    if (tickMs
> 60) tickMs -= 5;}
```

Cette boucle ajoute exactement 3 segments à la longueur du serpent, avec une vérification de limite pour éviter les dépassements de tableau. Une légère accélération optionnelle est appliquée après chaque séquence complétée.

Gestion des vies et réinitialisation

Détection de collision

La fonction ExitGame() vérifie les collisions avec les murs (x≤10, x≥70, y≤10, y≥30) et l'auto-collision du serpent avec son propre corps.

Réinitialisation complète

À chaque mort (si life ≥ 0) : length = 5, position par défaut (25,20), direction RIGHT, applesStreak = 0, et régénération de 3 nouvelles pommes.

Code de réinitialisation

```
if (life >= 0) {    length = 5; /* RESET LONGUEUR */    head.x = 25; head.y = 20;    bend_no = 0;
head.direction = RIGHT;    applesStreak = 0;    for (i = 0; i < 3; i++) {        foods[i].x = 0;
foods[i].y = 0;    }    Move();}
```

Ce bloc garantit un état de jeu propre après chaque mort, permettant au joueur de recommencer avec les paramètres initiaux tout en conservant son score total de pommes mangées.

Système de temporisation lissée

1

Variable tickMs

Durée d'une frame en millisecondes (initialisée à 110ms).

2

Fonction Sleep()

API Windows pour pause précise, évite le busy-wait inefficace.

3

Fonction Delay()

Appelle Sleep(tickMs) puis Score() pour mise à jour de l'affichage.

Cette approche garantit une vitesse homogène sur toutes les machines, contrairement aux anciennes méthodes de boucles vides qui dépendaient de la puissance du processeur.

Boucle principale de jeu

La fonction Move() constitue le cœur récursif du jeu, gérant le mouvement continu et la lecture des touches directionnelles.

Food()

Mise à jour des pommes et croissance

Delay()

Temporisation et affichage score

Boarder()

Rafraîchissement de l'écran

ExitGame()

Vérification des collisions

Direction

Mouvement selon la direction actuelle

Contrôles et directions

Touches fléchées

UP (72), DOWN (80), LEFT (75), RIGHT (77) - codes ASCII des touches directionnelles pour le contrôle du serpent.

Prévention demi-tour

Le code empêche le serpent de faire demi-tour instantanément (RIGHT→LEFT impossible), évitant l'auto-collision immédiate.

Mémorisation des coudes

Tableau bend[500] stocke chaque changement de direction pour dessiner correctement le corps du serpent.

Touche ESC

Permet de quitter le jeu à tout moment (code 27), avec nettoyage de l'écran avant sortie.

Fonctions de mouvement directionnelles

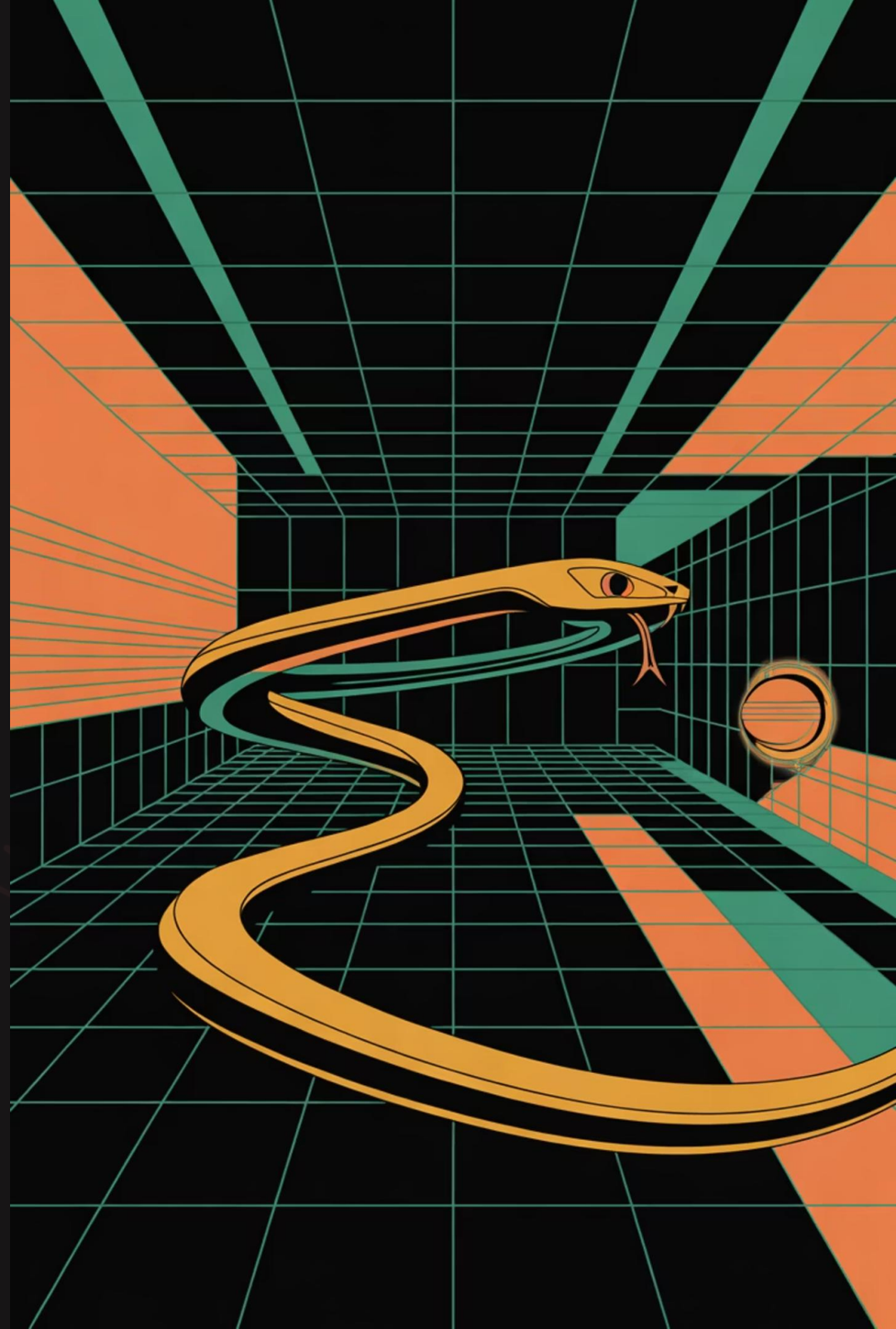
Quatre fonctions distinctes gèrent le déplacement dans chaque direction : Up(), Down(), Left(), Right(). Chacune dessine le serpent segment par segment.

Dessin progressif

Boucle for qui dessine chaque segment du corps, avec caractères spéciaux pour la tête (^, v, <, >) et le corps (**).

Gestion des coudes

Appel à Bend() pour dessiner les segments le long des changements de direction mémorisés.



Fonction Bend() - Dessin des coudes

Cette fonction complexe parcourt le tableau bend[ ] en sens inverse pour dessiner tous les segments du corps entre les points de changement de direction.

1

Parcours inverse

Commence à bend_no et descend jusqu'à 0, dessinant les segments entre chaque paire de coudes.

2

Segments verticaux

Si bend[i].x == bend[i-1].x, dessine verticalement selon la différence en y.

3

Segments horizontaux

Si bend[i].y == bend[i-1].y, dessine horizontalement selon la différence en x.

4

Limite de longueur

S'arrête dès que len == length pour ne dessiner que la longueur actuelle du serpent.

Affichage du score et des statistiques

5

Longueur initiale

Le serpent commence toujours avec 5 segments

3

Vies disponibles

Le joueur dispose de 3 vies au démarrage

110

Tick initial (ms)

Durée de frame au début du jeu

La fonction Score() affiche en temps réel : le score (length - 5), le nombre de vies restantes, et le total de pommes mangées. Ces informations sont positionnées en haut de l'écran via GotoXY().

Procédure de test complète

Lancement

Vérifier que 3 pommes 'F' sont visibles dans l'aire de jeu dès le départ.

Première pomme

Manger une pomme : la longueur reste à 5, applesStreak passe à 1.

Deuxième pomme

Manger la deuxième : longueur toujours 5, applesStreak = 2.

Troisième pomme

Manger la troisième : +3 segments d'un coup (longueur = 8), nouvelle séquence de 3 pommes apparaît.

Test de mort

Se cogner contre un mur : respawn avec longueur = 5, 3 nouvelles pommes générées.

Vitesse

Vérifier que la vitesse reste stable et homogène, avec légère accélération après chaque séquence.

Fonctionnalités additionnelles

Système de records

La fonction record() sauvegarde le nom du joueur, la date, et le score dans record.txt. Permet de consulter l'historique des parties précédentes.

Écran d'accueil

Print() affiche les instructions du jeu : utilisation des flèches, pause avec n'importe quelle touche, ESC pour quitter.

Écran de chargement

load() affiche une barre de progression animée "loading..." pour créer une transition au démarrage.

Tableau de synthèse technique

Exigence	Implémentation	Zones du code
3 pommes simultanées	Tableau foods[3], SpawnApple(), affichage des F	Déclarations, SpawnApple(), Boarder(), Food()
+3 après 3 pommes	Compteur applesStreak ; length += 3 quand == 3	Food(), Score(), variables globales
Reset longueur à 5	length = 5 et reset de la séquence	ExitGame()
Vitesse lissée	Sleep(tickMs) dans Delay()	tickMs (global), Delay()
Gestion des coudes	Tableau bend[500], fonction Bend()	Move(), Bend(), fonctions directionnelles
Sauvegarde scores	Écriture dans record.txt	record(), Scoreonly()

Conclusion et points clés

Objectifs atteints

Toutes les consignes sont implémentées : 3 pommes simultanées, croissance par séquence de +3, réinitialisation à 5 segments, et vitesse lissée.

Code structuré

Architecture claire avec séparation des responsabilités : génération, affichage, mouvement, collisions, et gestion d'état.

Expérience joueur

Gameplay fluide avec temporisation stable, système de vies, sauvegarde des scores, et interface utilisateur intuitive.