RAPPORT DE PROJET : TECH-TRIS

Développeurs : Adam et Anes

Date de rendu : 11 mai 2025

Langage : C

Environnement : Terminal POSIX

1. Introduction

Tech-Tris est une implémentation du jeu classique Tetris dans un environnement terminal. Ce projet s'inscrit dans une démarche d'approfondissement des connaissances en programmation C, en mettant l'accent sur la gestion des structures de données, les entrées/sorties, la gestion du temps et l'affichage en mode console.

L'objectif principal était de créer un jeu à la fois fonctionnel et agréable à utiliser, même dans l'environnement contraint d'un terminal. Nous avons également cherché à rendre le code modulaire, bien documenté et facilement extensible.

2. Architecture du projet

2.1 Organisation du code

Le projet a été structuré de manière modulaire, avec une séparation claire des responsabilités :

- **Module principal (`main.c`)** : Initialisation, boucle de jeu, gestion des entrées utilisateur
- **Module de jeu (`jeu.c`, `jeu.h`)** : Logique du jeu, gestion de la grille, vérification des collisions
- **Module des pièces (`piece.c`, `piece.h`)** : Définition et manipulation des pièces de Tetris
- **Module d'affichage (`display.c`, `display.h`)** : Rendu graphique dans le terminal
- **Module des scores (`score.c`, `score.h`)** : Gestion et persistance des scores

Cette organisation nous a permis de travailler efficacement en parallèle et de maintenir une base de code propre et lisible.

2.2 Structures de données principales

Plusieurs structures ont été conçues pour représenter les éléments du jeu :

- `Jeu` : Contient l'état complet du jeu (grille, score, niveau)
- `Piece` : Définit une pièce de Tetris (forme, symbole)
- `ScoreEleve` : Stocke les informations d'un score (nom, points, date)

3. Implémentation des fonctionnalités

3.1 Interface utilisateur et affichage

L'interface utilisateur a été soignée pour offrir une expérience visuelle agréable malgré les contraintes du terminal :

- Utilisation de séquences d'échappement ANSI pour les couleurs
- Encadrement de la grille et des informations avec des caractères Unicode
- Affichage en temps réel de la prochaine pièce
- Retour visuel des actions de l'utilisateur

Le module d'affichage a été conçu pour abstraire la complexité des séquences d'échappement et fournir des fonctions de haut niveau pour l'affichage du jeu.

3.2 Mécanique de jeu

La logique de jeu respecte les règles classiques du Tetris :

- Placement des pièces à la position et avec la rotation choisies
- Vérification des collisions et des limites de la grille
- Suppression des lignes complètes et décalage des lignes supérieures
- Augmentation progressive de la difficulté avec le niveau

Une attention particulière a été portée à la détection précise des collisions, notamment lors des rotations près des bords.

3.3 Gestion du temps et des entrées

Un aspect crucial du jeu est la gestion du temps :

- Utilisation de `setitimer` et de signaux pour limiter le temps de réponse du joueur
- Implémentation d'une fonction `lireAvecDelai` qui attend une entrée pendant un temps déterminé
- Remplacement de `usleep` par `nanosleep` pour une meilleure précision

Ces mécanismes permettent d'augmenter progressivement la difficulté en réduisant le temps disponible pour placer chaque pièce.

3.4 Système de score

Le système de score a été conçu pour être motivant et équilibré :

- Points de base attribués pour chaque ligne complétée
- Multiplicateur en fonction du nombre de lignes complétées simultanément
- Bonus de difficulté selon le niveau choisi
- Tableau des meilleurs scores persistant entre les sessions de jeu

4. Défis et solutions

4.1 Gestion non-bloquante des entrées

Un des défis majeurs était de gérer les entrées utilisateur avec une contrainte de temps. Nous avons résolu ce problème en :

- Utilisant des fonctions de bas niveau pour passer le terminal en mode non-canonique
- Implémentant un système de minuterie basé sur les signaux POSIX
- Combinant ces approches pour créer une fonction de lecture avec timeout

4.2 Rotation des pièces

La rotation des pièces a nécessité une attention particulière pour gérer correctement les cas limites. Notre solution consiste à :

- Implémenter les quatre rotations possibles (0°, 90°, 180°, 270°)
- Vérifier les collisions après rotation
- Détecter les parties significatives de la pièce pour un placement précis

4.3 Chargement dynamique des pièces

Pour rendre le jeu extensible, nous avons implémenté un système de chargement des pièces à partir de fichiers texte :

- Lecture des fichiers depuis un répertoire dédié
- Analyse du contenu pour créer les matrices représentant les pièces
- Système de fallback vers des pièces par défaut en cas d'échec du chargement

5. Répartition du travail

Ce projet a été réalisé dans une démarche de pair programming complète. Adam et Anes ont travaillé ensemble sur chaque partie du code, en partageant constamment leurs connaissances et en prenant les décisions de conception ensemble. Nous avons choisi cette approche pour :

- Améliorer la qualité du code grâce à une révision continue
- Partager les connaissances et compétences entre nous
- Assurer une cohérence stylistique et architecturale dans tout le code
- Renforcer notre compréhension mutuelle des différentes parties du projet

Cette méthode s'est révélée très efficace pour ce projet, nous permettant de résoudre rapidement les problèmes rencontrés et d'améliorer continuellement notre implémentation.

6. Tests et validation

Le jeu a été testé de manière approfondie sur différentes configurations :

- Tests unitaires informels pour les fonctions critiques
- Tests d'intégration des différents modules
- Tests de performance pour s'assurer d'une exécution fluide
- Tests de compatibilité sur différents terminaux

Nous avons également fait tester le jeu par des utilisateurs externes pour valider l'ergonomie et identifier les bugs potentiels.

7. Améliorations futures

Plusieurs pistes d'amélioration ont été identifiées pour des versions futures :

- Ajout d'un mode "fantôme" montrant où la pièce atterrira
- Implémentation d'un système de "hold" pour garder une pièce en réserve
- Ajout d'un mode multijoueur local
- Animations pour la suppression des lignes
- Support pour les terminaux sans couleur
- Système de sauvegarde/reprise de partie

8. Conclusion

Ce projet nous a permis d'approfondir notre maîtrise du langage C et de nous confronter à des problématiques variées : gestion fine des entrées/sorties, manipulation de structures de données complexes, programmation événementielle, etc.

La réalisation de Tech-Tris a été une expérience enrichissante qui démontre qu'il est possible de créer une expérience de jeu satisfaisante même dans un environnement aussi contraint qu'un terminal texte.

Nous sommes particulièrement satisfaits de la modularité du code et de l'expérience utilisateur obtenue, qui respecte l'esprit du Tetris original tout en ajoutant des touches personnelles.

9. Annexes

9.1 Instructions de compilation et d'exécution

bash

```
# Compilation
gcc -o tech-tris main.c jeu.c piece.c display.c score.c -std=c99

# Exécution
./tech-tris
```

9.2 Format des fichiers de pièces

Chaque pièce est définie dans un fichier texte de format 5×5 où :

- Les espaces représentent les cellules vides
- Tout autre caractère représente une cellule occupée

Exemple pour une pièce en forme de L :

```
X
X
XX
```

9.3 Références

- "The C Programming Language" par Kernighan et Ritchie
- Documentation POSIX pour les fonctions système utilisées
- Spécification des séquences d'échappement ANSI pour les terminaux