Rapport de Projet : Space Invaders

Implémentation MVC en C avec SDL3 et Ncurses

Auteurs :
Yaël MUSELET DUMONT
Théo HECQUET

Cadre du projet :
Programmation Avancée en C

03 January 2026

1. Introduction

Ce projet a pour objectif la conception et l'implémentation d'un clone du célèbre jeu d'arcade **Space Invaders**, développé intégralement en langage C. La contrainte majeure et l'originalité de ce projet résident dans son architecture logicielle : le jeu doit être jouable à travers deux interfaces radicalement différentes — une interface graphique moderne (SDL3) et une interface textuelle rétro (Ncurses) — tout en partageant le même noyau logique.

Ce rapport détaille les choix architecturaux, les défis techniques rencontrés (notamment la gestion de la mémoire et de l'audio), ainsi que les méthodes de validation mises en œuvre pour garantir la robustesse du programme.

2. Architecture Logicielle (MVC)

Pour répondre à la contrainte de dualité d'affichage, nous avons adopté le patron de conception Modèle-Vue-Contrôleur (MVC). Cette approche permet de découpler strictement la logique du jeu de sa représentation.

2.1. Le Modèle (`model.c`)

Le Modèle est le cœur du jeu. Il contient les structures de données définies dans `model.h` (`GameState`, `Entity`, `Player`, `Enemy`) et les fonctions de manipulation de l'état du jeu.

- **Responsabilité :** Calculer les déplacements, détecter les collisions (AABB), gérer le score et les vies.
- **Indépendance :** Le modèle ne contient aucun pointeur vers SDL ou Ncurses. Il manipule des coordonnées abstraites.

2.2. Les Vues (`view_sdl.c` et `view_ncurses.c`)

Nous avons implémenté deux modules de vue distincts qui ne communiquent qu'en lecture avec le Modèle.

- Vue SDL3 : Utilise l'accélération matérielle pour afficher des sprites (`.png`) et gère le système audio (`SDL_Mixer`). Elle offre une expérience fluide à 60 FPS en plein écran.
- Vue Ncurses : Utilise la bibliothèque standard de terminal pour afficher le jeu en caractères ASCII. Elle permet de jouer via SSH ou sur des machines sans interface graphique.

2.3. Le Contrôleur (`controller.c`)

Ce module sert d'adaptateur. Il intercepte les événements bruts (touches clavier, clics souris) provenant de la librairie active (SDL ou Ncurses) et les traduit en commandes sémantiques pour le modèle (ex: `player_move`, `player_shoot`).

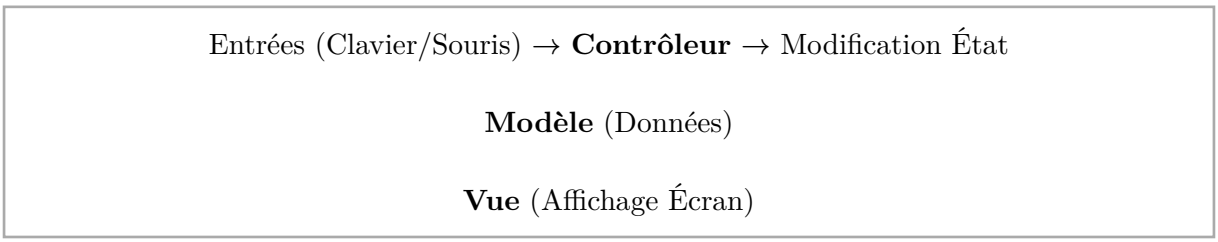

Fig. 1. – Flux de données dans notre architecture MVC

3. Implémentation Technique

3.1. Boucle de Jeu et « Timestep »

Un défi majeur lors de la création d'un jeu compatible SDL (60Hz) et Ncurses (vitesse variable) est la synchronisation. Si la logique du jeu est liée au taux de rafraîchissement, le jeu serait injouable en mode terminal (trop rapide).

Nous avons implémenté une boucle de jeu à pas de temps fixe (Fixed Timestep). Le temps écoulé est accumulé dans une variable. La mise à jour logique (`update_model`) n'est déclenchée que lorsque l'accumulateur atteint un seuil défini (ex: 16ms pour 60 updates/sec), indépendamment de la vitesse d'affichage de la vue.

3.2. Gestion de l'Audio avec SDL3

L'intégration de l'audio a nécessité une attention particulière, SDL3 étant encore en phase de développement actif.

- **Problème de boucle :** La valeur `-1` pour boucler la musique infiniment provoquait parfois des arrêts inopinés.
- **Solution :** Nous avons opté pour une boucle explicite de `100 000` répétitions lors de l'appel à `MIX_PlayTrack`, garantissant une lecture continue.
- **Conflits Pause/Lecture :** Lors de la navigation entre les menus (Accueil, Jeu, Game Over), la relance de la musique causait des conflits. Nous avons résolu cela en forçant un `MIX_HaltTrack` (arrêt total) avant chaque nouvelle lecture, plutôt que de gérer des états de pause complexes.

3.3. Gestion de la Mémoire et Nettoyage

Le C exige une gestion manuelle de la mémoire. Nous avons utilisé des outils dynamiques pour assurer l'absence de fuites.

- **Problème de Segfault à la fermeture :** Initialement, le jeu plantait en quittant. L'analyse a révélé que nous détruisions le contexte audio (`MIX_DestroyAudio`) avant les pistes (`MIX_DestroyTrack`). SDL essayait d'arrêter une piste dont la source n'existait plus.
- **Correction :** La fonction `cleanup_audio` a été réécrite pour respecter l'ordre strict de destruction : Tracks → Chunks → Device.

4. Automatisation et Qualité

Pour garantir la pérennité du projet, nous avons mis en place un environnement de développement robuste.

4.1. Makefile Automatisé

Le fichier `Makefile` ne se contente pas de compiler. Il gère l'installation complète des dépendances :

- Détection de l'OS (Ubuntu vs Fedora).
- Téléchargement et compilation de SDL3 depuis les sources si nécessaire.
- Règles séparées pour `make run_sdl`, `make run_nc` et `make test`.