Rapport de Projet : Space Invaders en C (MVC)

Université d'Artois - Licence Informatique L3
Cours : C Avancé
Groupe 1 : Julien Delcourt & Gwenaëlle Roussel
Date : Janvier 2026

Table des matières

1. Introduction

Contexte et objectifs

Ce projet consiste en l'implémentation d'un clone du jeu d'arcade classique Space Invaders en langage C, respectant strictement le patron de conception MVC (Modèle-Vue-Contrôleur). L'objectif principal est de créer une application modulaire capable de fonctionner avec deux interfaces utilisateur distinctes : une interface texte utilisant ncurses et une interface graphique basée sur SDL3, tout en maintenant une séparation claire entre la logique métier et l'affichage.

2. Architecture et organisation du projet

Structure du patron MVC L'architecture repose sur une séparation stricte en trois composants, conformément au modèle MVC. Cette organisation garantit la réutilisabilité du code et facilite la maintenance.

Le Modèle (model.c/h)

Le modèle constitue le cœur du système. Il encapsule toute la logique métier du jeu et gère l'état global via la structure EtatJeu. Cette structure contient les positions des entités (vaisseau, ennemis, projectiles), les statistiques de jeu (score, vies, niveau) et les paramètres de difficulté.

Le modèle est totalement agnostique des bibliothèques d'affichage. Il ne contient aucune référence à ncurses ou SDL, ce qui permet de le réutiliser avec n'importe quelle interface.

Les Vues (view_console.c/h et view_sdl.c/h)

Les vues sont responsables uniquement du rendu visuel. Elles interrogent le modèle via des accesseurs (getters) pour obtenir les informations nécessaires à l'affichage, mais ne modifient jamais directement l'état du jeu.

La vue console (view_console.c) utilise ncurses pour transformer la grille logique en représentation textuelle. Chaque entité du jeu (vaisseau, ennemi, projectile) est représentée par un caractère ASCII avec colorisation si le terminal le supporte. La vue SDL (view_sdl.c) est actuellement un stub léger qui sera développé ultérieurement pour fournir un rendu graphique complet.

Le Contrôleur (controller.c/h)

Le contrôleur fait le lien entre les entrées utilisateur et le modèle. Il définit un ensemble de commandes.

Les vues capturent les événements clavier bruts (touches 'a'/'d' ou flèches en ncurses, événements SDL_KEYDOWN en SDL) et appellent directement la commande appropriée. Le contrôleur reçoit ces commandes via un simple switch et invoque les fonctions appropriées du modèle, garantissant ainsi que la logique de traitement des entrées est identique quelle que soit l'interface utilisée.

Organisation des fichiers.

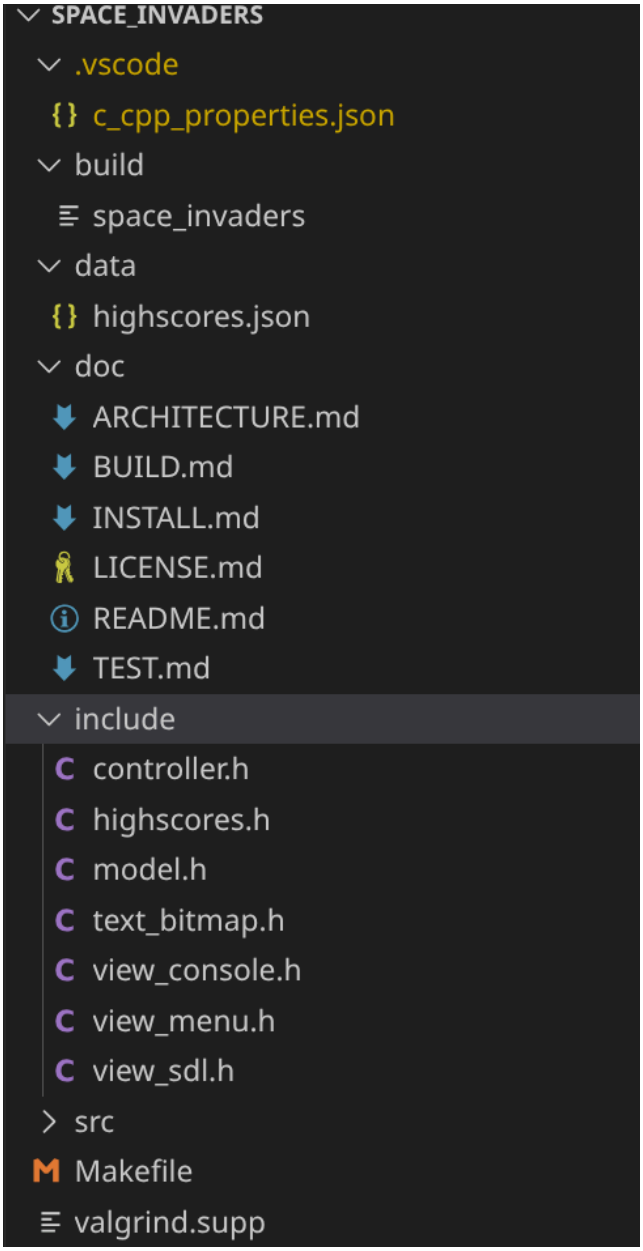
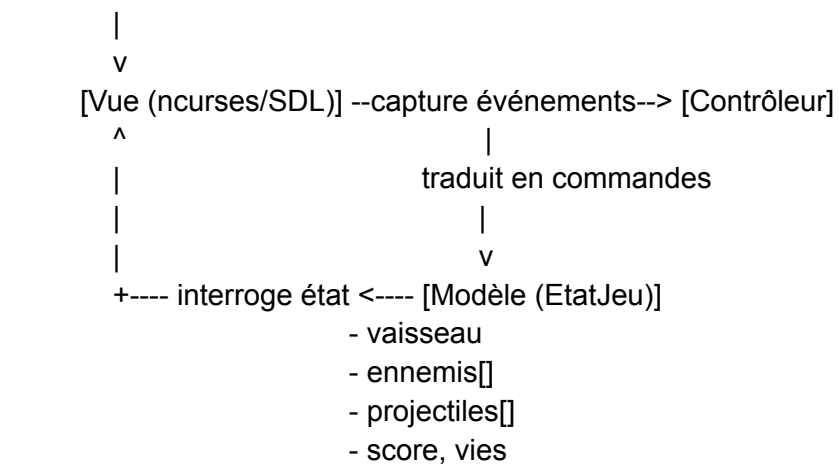


Diagramme de flux



3. Choix Techniques : Le système d'Entités

Pour rendre le code plus intelligent, nous avons décidé que tout ce qui est à l'écran est une "Entité".

3.1. La structure commune

Chaque objet (Vaisseau, Alien, Tir, Barricade) utilise la même base de données. C'est ce qu'on appelle une Unification :Vie : Un Alien a 1 PV, le Vaisseau a 3 PV, et la Barricade a 3 PV.

- Position : Toutes les entités ont des coordonnées X et Y.
- État : Un drapeau "Vivant/Mort" pour savoir s'il faut encore l'afficher.

3.2. Les Barricades (Le bouclier destructible)

C'est le choix technique majeur du projet :

- Fonctionnement : La barricade est une entité qui ne bouge pas.
- Résistance : Elle possède 3 points de vie.
- Effet : Chaque fois qu'un projectile (joueur ou ennemi) la touche, elle perd 1 PV. Au bout de 3 coups, elle est supprimée du plateau. Cela protège le joueur tout en étant limité.

3.3. Récompense (Le système d'XP)

- Score : Chaque ennemi détruit rapporte 10 points d'XP.
- Progression : Ce système est centralisé dans le Modèle. Dès qu'un ennemi passe à "Mort", le compteur d'XP du joueur augmente automatiquement.

4. Difficultés rencontrées et solutions

4.1 La fin de ncurses sur Windows (Janvier 2026)

- **Problème :** Les dépendances ncurses ont été totalement supprimées des environnements Windows modernes, rendant la vue console impossible à compiler.
- **Solution :** Avoir un vrai système d'exploitation sous Linux.

4.2. L'apprentissage accéléré de SDL3

- **Problème :** Passer de ncurses à SDL3 demande un temps d'apprentissage important (gestion des fenêtres, des textures, …).
- **Solution :** Allez en cours, particulièrement en TP.

4.3. Le rendu de texte "fait maison"

- **Problème :** SDL3 ne possède aucun module natif pour écrire du texte. Sans bibliothèques externes, le jeu ne pouvait afficher ni le score, ni les vies.
- **Solution :** Nous avons **recodé notre propre système d'affichage dans un fichier**. Chaque lettre a été dessinée manuellement avec des fonctions.

5. Tests et validation

5.1 Tests de robustesse mémoire (Valgrind)

Nous avons exécuté le jeu sous Valgrind pour détecter les fuites mémoire et les accès invalides, en utilisant avec le Makefile avec la version console et SDL.

Le rapport confirme l'absence d'importantes fuites mémoire. Toutes les allocations (structure EtatJeu et buffer d'affichage console) sont correctement libérées à la fin du programme.

```
gwen@gwenDebian:~/Téléchargements/space_invaders$ make valgrind-sdl
=== Valgrind Memory Check (SDL) ===
Note: Fermez la fenêtre pour voir le rapport
Les fuites SDL 'still reachable' sont normales
valgrind --leak-check=full --show-leak-kinds=definite,possible --suppressions=valgrind.supp
--track-origins=yes build/space_invaders --view=sdl
==216714== Memcheck, a memory error detector
==216714== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==216714== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==216714== Command: build/space_invaders --view=sdl
==216714==
==216714==
==216714== HEAP SUMMARY:
==216714==     in use at exit: 337,019 bytes in 3,560 blocks
==216714==   total heap usage: 185,818 allocs, 182,258 frees, 54,277,449 bytes allocated
==216714==
==216714== LEAK SUMMARY:
==216714==    definitely lost: 0 bytes in 0 blocks
==216714==    indirectly lost: 0 bytes in 0 blocks
==216714==      possibly lost: 0 bytes in 0 blocks
==216714==    still reachable: 67,784 bytes in 450 blocks
==216714==         suppressed: 269,235 bytes in 3,110 blocks
==216714== Reachable blocks (those to which a pointer was found) are not shown.
==216714== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==216714==
==216714== For lists of detected and suppressed errors, rerun with: -s
==216714== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
gwen@gwenDebian:~/Téléchargements/space_invaders$
```

```
gwen@gwenDebian:~/Téléchargements/space_invaders$ make valgrind-console
=== Valgrind Memory Check (Console) ===
Note: Appuyez sur 'q' pour quitter et voir le rapport
Les fuites SDL/ncurses 'still reachable' sont normales
valgrind --leak-check=full --show-leak-kinds=definite,possible --suppressions=valgrind.supp
--track-origins=yes build/space_invaders --view=console
==216532== Memcheck, a memory error detector
==216532== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==216532== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==216532== Command: build/space_invaders --view=console
==216532==
==216532==
==216532== HEAP SUMMARY:
==216532==     in use at exit: 875,434 bytes in 492 blocks
==216532==   total heap usage: 537 allocs, 45 frees, 893,476 bytes allocated
==216532==
==216532== LEAK SUMMARY:
==216532==    definitely lost: 0 bytes in 0 blocks
==216532==    indirectly lost: 0 bytes in 0 blocks
==216532==      possibly lost: 0 bytes in 0 blocks
==216532==    still reachable: 5,928 bytes in 247 blocks
==216532==         suppressed: 869,506 bytes in 245 blocks
==216532== Reachable blocks (those to which a pointer was found) are not shown.
==216532== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==216532==
==216532== For lists of detected and suppressed errors, rerun with: -s
==216532== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 9 from 9)
```

5.2 Tests manuel : Nous avons testé manuellement le jeu en SDL et console pour vérifier les potentiels problèmes.

6. Conclusion

Ce projet a permis de mettre en pratique de nombreux concepts avancés de la programmation en C : gestion manuelle de la mémoire, encapsulation par types opaques, séparation des responsabilités via MVC, et intégration de bibliothèques tierces (ncurses pour l'instant, SDL3 à venir).

Ce projet a été l'occasion de constater l'importance d'une architecture bien pensée dès le départ. Le respect strict du modèle MVC, bien que contraignant initialement, s'est révélé payant lors de l'implémentation de la seconde interface. La discipline imposée par le C (gestion manuelle de la mémoire, absence d'objets) nous a forcés à réfléchir en profondeur à l'organisation du code et à l'encapsulation des données.