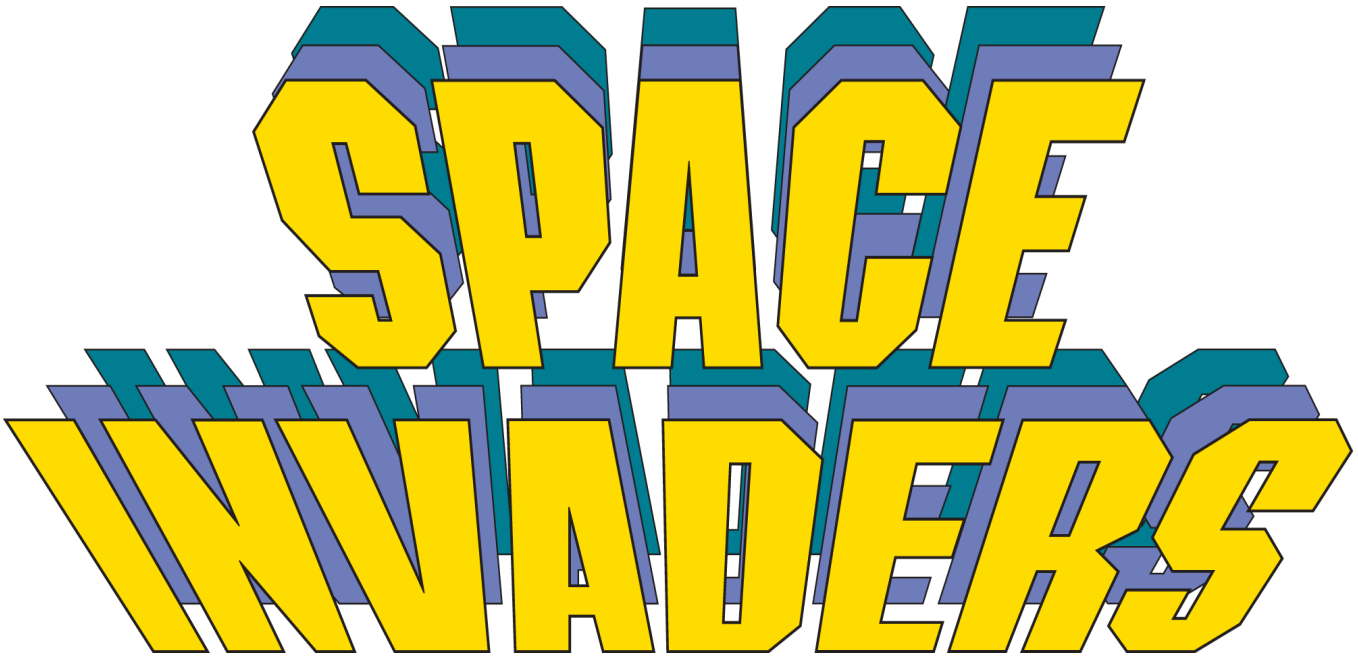


- **Membres du groupe :**

Etudiante 01	MEZRAGUE	Yasmine
Etudiante 02	BENALI	Meriem

- **Niveau :** L3 informatique.

- **Établissement :** Université d'Artois.

- **Enseignant du module C avance :**

Jean-Marie Lagniez.

1. Introduction :

Ce rapport présente le développement d'un jeu Space Invaders en langage C, structuré selon le modèle MVC (Modèle–Vue–Contrôleur) afin de séparer clairement la logique du jeu, l'affichage et la gestion des interactions utilisateur. Le programme est compatible avec deux interfaces d'affichage distinctes : une interface texte utilisant **ncurses** et une interface graphique reposant sur **SDL3**.

Le document décrit l'organisation générale du code, le découpage en modules, les choix techniques retenus ainsi que les tests effectués pour assurer la robustesse et la fiabilité du logiciel.

2. Objectifs du projet :

- Séparer clairement la logique du jeu, l'affichage et la gestion des entrées utilisateur.
- Assurer la réutilisabilité du modèle de jeu pour les deux interfaces.
- Implémenter les principales mécaniques du jeu : déplacement du vaisseau, gestion des ennemis, tirs et collisions, score et vies.
- Développer un code robuste, modulaire et facilement maintenable.
- Gérer correctement la mémoire et éviter les fuites, notamment en utilisant Valgrind.

3. Contraintes techniques :

Contrainte	Description
Langage C	Le projet est entièrement développé en C, sans recours au C++ ou à la programmation orientée objet.
Architecture MVC	Séparation stricte entre le modèle (logique du jeu), la vue (affichage) et le contrôleur (gestion des entrées).
Interfaces d'affichage	Le jeu est conçu pour fonctionner avec deux interfaces interchangeables : ncurses pour l'affichage texte dans le terminal et SDL3 pour l'affichage graphique avec sprites et animations simples
Portabilité Linux	Le code doit être compilable et exécutable sur Linux.

Boucle de jeu et framerate	La boucle principale doit assurer un comportement identique entre les deux vues grâce à un timestep fixe ou semi-fixe.
Modularité	Le code est découpé en modules distincts (modèle, contrôleur, vues, utilitaires), facilitant la maintenance et la réutilisation.
Robustesse et mémoire	Gestion propre des erreurs, des allocations mémoire et absence de fuite mémoire (à vérifier avec Valgrind).
Construction	Fourniture d'un Makefile permettant de compiler, exécuter et nettoyer le projet facilement.

4. Architecture générale :

Le projet est structuré selon une architecture **MVC** (Modèle–Vue–Contrôleur) afin de séparer clairement la logique du jeu, l'affichage et la gestion des interactions utilisateur.

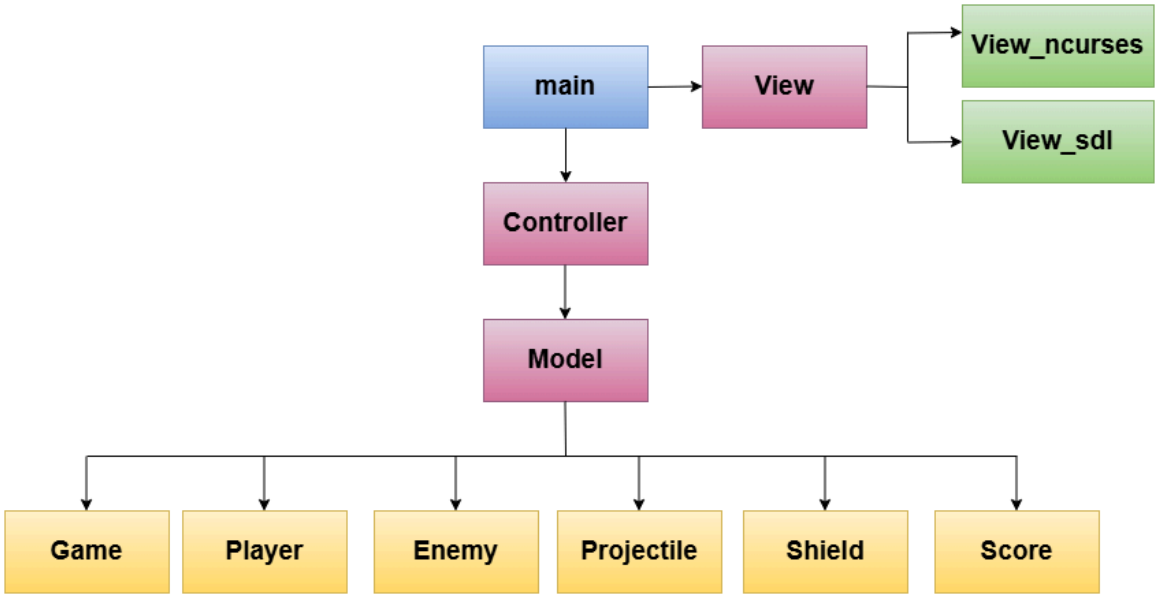
- **Modèle (src/model/) :**
  Le modèle contient toute la logique du jeu et représente son état interne. Il gère les entités du jeu telles que le joueur, les ennemis, les projectiles et les boucliers, ainsi que le score et les niveaux. Chaque composant est isolé dans des fichiers spécifiques (player.c/h, enemies.c/h, projectiles.c/h, shields.c/h, score.c/h) pour assurer la modularité.

- **Contrôleur (src/controller/) :**
  Le contrôleur sert d'intermédiaire entre les entrées utilisateur et le modèle. Il traduit les actions de l'utilisateur (déplacement, tir, pause) en commandes abstraites, puis met à jour le modèle en conséquence. Il assure également la synchronisation entre la boucle de jeu et le modèle, tout en restant indépendant des vues utilisées (ncurses ou SDL3).

- **Vue (src/view/) :**
  La vue est responsable de l'affichage des informations et du rendu du jeu, sans contenir de logique de jeu. Elle repose sur une interface abstraite définie dans **view.h**, qui sert de pont entre le modèle et les deux implémentations concrètes : **view_ncurses**, pour un affichage texte lisible dans le terminal, et **view_sdl**, pour un affichage graphique avec sprites, animations et barre d'information. Cette abstraction permet au contrôleur et au modèle de rester totalement indépendants de l'interface utilisée, assurant ainsi la réutilisation, la maintenabilité et la modularité du code.

- **Utils (src/utils/) :**
  Le dossier utils contient des fonctions utilitaires réutilisables par le modèle, le contrôleur ou les vues. Il inclut notamment la gestion du temps et des temporisations dans la boucle de jeu via timer.c/h, ainsi que la lecture et l'écriture de fichiers pour la sauvegarde de scores ou la configuration grâce à file_io.c/h. Ces fonctions centralisent les opérations communes, facilitent la maintenance et évitent la duplication de code dans le projet.

5. Diagramme d'architecture :



6. Difficultés rencontrées :

- **Installation et configuration de SDL3 :**
  La mise en place de la bibliothèque SDL3 sur Linux a été complexe, notamment à cause de la compatibilité avec différents drivers vidéo et des versions des dépendances. Cela a nécessité plusieurs tests et ajustements pour s'assurer que la SDL3 s'initialise correctement et que les fenêtres se créent sans erreur.

- **Respect strict du modèle MVC :**
  Maintenir une séparation nette entre le modèle, le contrôleur et les vues a demandé une attention particulière, surtout pour l'interface abstraite view.h, afin de pouvoir basculer facilement entre ncurses et SDL3 sans introduire de dépendances croisées.