Introduction

Space Invaders est un jeu d'arcade classique où le joueur contrôle un vaisseau horizontal en bas de l'écran et doit détruire des vagues d'ennemis qui descendent (https://fr.wikipedia.org/wiki/Space_Invaders).
Dans ce projet, vous développerez une version de *Space Invaders* en langage C pouvant fonctionner sur deux interfaces : une interface texte (ncurses) et une interface graphique (SDL). L'architecture du projet doit respecter le modèle MVC (Modèle–Vue–Contrôleur) pour séparer clairement la logique du jeu, l'affichage et la gestion des entrées.

Objectif

Concevoir et implémenter un Space Invaders jouable en C, avec deux vues (ncurses + SDL) interchangeables au lancement, en respectant le modèle MVC. Le projet doit être modulaire, documenté et exécutable sur Linux.

Fonctionnalités requises

Modèle

- Gestion de l'état du jeu : positions des ennemis, du vaisseau, projectiles, boucliers, scores, vies, niveau courant.
- Logique de déplacement des ennemis (mouvement en groupe, changement de direction, accélération selon le nombre d'ennemis restants).
- Gestion des tirs : tirs du joueur et tirs ennemis, collisions (projectile ↔ ennemi, projectile ↔ vaisseau, projectile ↔ bouclier).
- Gestion des niveaux : génération de vagues, progression de la difficulté.
- Système de score et vies.
- Sauvegarde/chargement optionnel des meilleurs scores (fichier local).

Le modèle **ne doit pas** dépendre des vues ou du contrôleur (pas d'appels à ncurses ou SDL depuis le modèle).

Vue — ncurses

- Affichage texte du plateau de jeu (vaisseau, ennemis, projectiles, boucliers, score, vies, niveau).
- Représentation claire et lisible dans le terminal, gestion de la taille minimale du terminal.
- Indication d'états (pause, game over, affichage du menu).

Vue — SDL

- Affichage graphique avec SDL3 offrant : sprite simple pour le vaisseau, ennemis et projectiles, barre d'information (score/vies/niveau).
- Animations basiques (déplacement ennemi, explosion simple).
- Option : affichage plein écran ou fenêtre redimensionnable.

Contrôleur

- Traitement des entrées clavier (gauche/droite, tirer, pause, changer d'interface si demandé au démarrage).
- Interface unique du contrôleur vers le modèle (ex : commandes abstraites : MOVE_LEFT, MOVE_RIGHT, SHOOT, PAUSE).
- Gestion des événements provenant de ncurses ou SDL via un adaptateur qui traduit événements UI → commandes contrôleur.

Exigences techniques

- **Basculer au lancement** : le jeu doit proposer (via option en ligne de commande ou menu au démarrage) de choisir la vue (ncurses ou SDL). Le basculement à chaud pendant l'exécution n'est pas requis, mais le modèle doit être réutilisable par les deux vues.
- **MVC strict** : le modèle ne dépend d'aucune bibliothèque d'affichage, la Vue ne contient pas la logique du jeu, et le Contrôleur orchestre les interactions.
- **Langage** : C (ISO C99 ou plus récent). Pas d'utilisation d'OO en C++.
- **Construction** : fournir un Makefile (ou CMake) pour compiler sur Linux (Debian/Ubuntu). Les dépendances (ncurses, SDL3) doivent être documentées.
- **Framerate & boucle de jeu** : boucle de jeu avec timestep fixe ou semi-fixe pour assurer comportement identique entre les vues.
- **Modularité** : code découpé en modules (ex : model.c/h, controller.c/h, view_ncurses.c/h, view_sdl.c/h, utils/), et commentaires explicatifs.
- **Robustesse** : gestion propre des erreurs (allocation, fichier, initialisation SDL/ncurses).
- **Mémoire** : pas de fuite mémoire ; fournir une preuve via valgrind (rapport court).
- **Licence** : préciser la licence du code (MIT/Unlicense, etc.) dans le dépôt.

Bonus (facultatif)

- Effets sonores et musique en SDL (utiliser SDL_mixer).
- Animations avancées et sprites.
- Niveaux multiples et boss.
- IA améliorée (ennemis qui ciblent le joueur ou patterns différents).
- Mode deux joueurs en local (partage clavier) ou en réseau (socket) — *attention : le réseau augmente fortement la complexité*.
- Sauvegarde persistante du classement (fichier JSON ou texte).
- Interface de configuration (réglage sensibilité, keybindings).
- Mode « directions relatives » pour contrôles alternatifs (voir ton idée moto/tron).
- Interface automatisée de tests unitaires sur la logique (scripts de tests pour le modèle).
- Support des manettes via SDL.

Livrables

1. **Code source complet** organisé en répertoires (src/, include/, assets/, doc/, build/).
2. **Makefile** (ou CMakeLists.txt) avec cibles : make, make run-ncurses, make run-sdl, make clean, make valgrind.
3. **README** détaillé : compilation, dépendances (versions recommandées), exécution, commandes clavier, description de l'architecture MVC.
4. **Rapport écrit (~4 pages)** expliquant : architecture, diagramme simple (modules), décisions de conception, difficultés, tests réalisés.
5. **Vidéo (~10 minutes)** présentant l'architecture, choix techniques, points critiques, puis une démo montrant les deux interfaces (ncurses + SDL).
6. **Rapport Valgrind** (preuve d'absence de fuites majeures).
7. **(Facultatif)** fichiers de configuration, scripts d'installation, et scores persistants.

Vous êtes autorisés à utiliser un LLM (comme ChatGPT, Copilot, etc.) pour vous assister dans la réalisation du projet.
En contrepartie, j'attends un niveau d'exigence élevé : le code, la documentation, la vidéo de présentation et le rapport devront refléter une réelle compréhension du travail réalisé, une architecture claire et une qualité de production professionnelle.