Projet d'implémentation : Snake[1] (sous MLV)

1 Organisation et Planning

Le projet doit être réalisé en **binôme** (date limite pour la communication du groupe le 4 novembre, sur moodle). Vous devez implémenter en C un jeu du type Snake, puis le déposer sur moodle avant le lundi 5 janvier 2026 18h00 (les soutenances auront lieu le mardi 6 janvier).

2 Livrables

Le dépôt final se fera sous la forme d'une archive `NOM1_PRENOM1_NOM2_PRENOM2.tar.gz` contenant **uniquement** :

- L'ensemble du code source (fichiers `.c`, `.h`).
- Le `Makefile` permettant de compiler **silencieusement** avec la commande `make`. Votre projet sera compilé sous vos yeux en salle de TP.
- Une notice `README` expliquant : compilation, exécution, sauvegardes (slots), et la structure des fichiers de sauvegarde.
- Éventuellement des images/ressources nécessaires (dans un dossier `ressources/`).

3 Contraintes techniques obligatoires

En plus des règles de programmation imposées en cours/TD/TP, vous devez suivre les contraintes suivantes.

- Le projet doit être écrit en **C** en utilisant la bibliothèque graphique **MLV** (installation disponible sur les postes de TP). L'utilisation de MLV est **obligatoire**.
- Le projet doit compiler silencieusement avec la commande `make` dans le répertoire racine du projet (sans arguments).
- Votre projet doit être portable : évitez les chemins absolus, par exemple, utilisez `./ressources/` pour les ressources.
- Le jeu doit être robuste. Par exemple, une saisie erronée ou un fichier absent ne doit pas faire planter le programme.
- Gérer le rafraîchissement (FPS) correctement et proprement : limiter le nombre d'appels coûteux à `MLV_actualize_window()`. Pour cela, vous devez utiliser une boucle principale de jeu en suivant la structure fournie dans la Section 5.

4 Fonctionnalités obligatoires

Les éléments suivants doivent être obligatoirement implémentés :

1. **Contrôles.** Le joueur doit contrôler le serpent avec les touches directionnelles. Gérer les appuis prolongés et éviter les changements de direction illogiques (exemple, que se passe-t-il en cas de demi-tour instantané ?).

2. **Pause et sauvegarde.** Une fonctionnalité de mise en `pause` du jeu doit permettre d'interrompre la partie et d'accéder au menu de pause. Le joueur doit pouvoir **sauvegarder** l'état courant de la partie dans un des 4 *slots* de sauvegarde (obligatoire) pour pouvoir charger ultérieurement une sauvegarde.

1. https://fr.wikipedia.org/wiki/Snake_(genre_de_jeu_vidéo)

3. **Scores.** Conserver et afficher les 10 meilleurs scores (persistants entre exécutions). À vous de définir la formule de score (fruits, temps, longueur, ...).
4. **Menu principal.** Le menu doit proposer au minimum :
   — Nouvelle partie,
   — Charger partie,
   — Sauvegarder partie (depuis la pause),
   — Meilleurs scores,
   — Quitter.
5. **Règles du Snake** à implémenter obligatoirement.
   — la tête meurt au contact du corps (partie perdue) ;
   — la tête réapparaît de l'autre côté de l'écran s'il n'y a pas de mur sur les bords.
6. **Organisation du code en modules.** N'oubliez de créer différents modules associés à une tâche précise. De même, préférez les fonctions courtes avec un rôle précis.
7. **Au moins 2 fonctionnalités "pour aller plus loin".** Toute initiative justifiée sera valorisée lors de la notation.
   Suggestions (liste non exhaustive) :
   — mode 2 joueurs sur le même écran ;
   — niveaux avec murs/obstacles, labyrinthes ;
   — objets spéciaux (accélérateurs, ralentisseurs, invincibilité temporaire, etc) ;
   — sauvegardes en format binaire ;
   — affichage d'un chrono et prise en compte du temps dans le score.

5 Gestion du temps réel et boucle de jeu

La boucle principale (boucle de jeu) doit :

1. Récupérer le temps au début de l'image,
2. Traiter les événements utilisateur (dans l'idéal, un seule évènement par image),
3. Mettre à jour les positions (logique du jeu),
4. Détecter et résoudre les collisions,
5. Afficher l'image et limiter le tempo pour atteindre la cadence désirée en attendant si nécessaire.

Voici à quoi doit ressembler votre boucle de jeu (pensez à implémenter des fonctions pour intermédiaires pour que cette boucle ne fasse que quelques lignes.

```
/* Boucle de jeu - image par image ... */
while (! quitter ){
    /* Récuperation du temps en nanosecondes au debut de l'image */
    clock_gettime(CLOCK_REALTIME, &debut );
    /* Affichage de l'image courante  */
    /* puis Récuperation d'au plus un evenement de clavier par image */
    /* puis résolution des évènements */
    /* puis déplacement des objets */
    /* puis résolution des collisions */
    /* Récuperation du temps en nanosecondes à la fin de l'image */
    clock_gettime(CLOCK_REALTIME, &fin );
    /*  puis calcul du temps passé sur l'image courante */
    /* puis on attend si l'image a été trop rapide */
}
```

6 Critères d'évaluation (liste non exhaustive)

La notation tiendra compte des éléments suivants :
— votre capacité à expliquer votre code et à le modifier à la volée ;
— respect des contraintes et fonctionnalités obligatoires ;
— qualité du code : lisibilité, modularité, commentaires, structure du projet ;
— robustesse : gestion des erreurs, stabilité à l'exécution ;
— performance et gestion correcte du FPS ;
— qualité de l'interface (menus, retours utilisateur) ;
— fonctionnalités additionnelles et originalité.