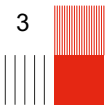
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

PROJET PMND : JEU EII SURVIVOR

L'HOSTIS Marion ; NGUYEN DUC BINH ; BOU HAMDAN Abdallah; ZONGO Yannis ; DURIF Theo

Université de Rennes

SOMMAIRE



INTRODUCTION

Les périodes d'examens comptent parmi les moments les plus intenses de la vie étudiante. Véritable course de fond, elles mêlent stress, fatigue et besoin constant de concentration. Dans ce contexte, trouver le bon équilibre pour maximiser ses performances est un réel défi. C'est précisément cet univers que nous avons choisi de mettre en scène à travers un jeu de simulation inédit, jouable grâce à une manette spécifiquement conçue et construite pour le projet.

C'est le défi que nous avons choisi de relever avec ce projet de PMND, et nous l'avons appelé "*EII Survivor*".

En langage C, et à l'aide de la bibliothèque SDL2, nous avons conçu l'interface du jeu. A partir des outils qu'offrent le microcontrôleur STM32, un clavier 4x3 et un joystick, nous avons développé la manette.

Il s'agit d'un jeu de plateforme dans lequel le joueur incarne un étudiant traversant sa période d'examens. À travers une série de mini-jeux, il accumule des points et des malus qui impactent directement ses statistiques physiques et mentales : son niveau de stress, sa fatigue, ainsi que ses connaissances. Au terme de l'aventure, une fois tous les mini-jeux effectués, le cumul de tous les scores est effectué et le score final obtenu. S'il est supérieur à la moyenne fixée, alors le joueur est déclaré vainqueur.

Au-delà de la créativité, ce projet nous a confronté à des problématiques concrètes de développement logiciel et matériel, tels que la gestion de mémoire, l'organisation Modèle-Vue-Contrôleur (MVC), le travail collaboratif sur GitLab, la liaison périphérique-logiciel, le rendu graphique en temps réel ainsi que la gestion de projet. Il représente pour nous une première expérience complète en conception d'un système interactif, du game design jusqu'à l'implémentation.

Ce compte rendu retrace les étapes de cette conception, les choix techniques que nous avons effectués, ainsi que les difficultés que nous avons rencontrées.

I. ORGANISATION ET STRUCTURE DU PROJET

Architecture logicielle (MVC) et gestion des données

Pour garantir la maintenabilité et la modularité de l'application, le développement logiciel s'appuie sur le patron de conception Modèle-Vue-Contrôleur (MVC). L'ensemble est orchestré par une structure centrale, Application, qui centralise le render, la police, la page active et les données de la partie. La transition entre les écrans est sécurisée par la fonction “*change_page*”, qui libère la mémoire de la page précédente avant d'initialiser la suivante.

- **Le Modèle :** Il regroupe toute la logique du jeu. La structure Player encapsule les statistiques du joueur (fatigue, stress, connaissances) et ses résultats aux mini-jeux (*MG_Results*). La structure “*GameStats*” gère le *timer* global, la difficulté et les déclencheurs d'événements (*clinic_trigger, sleep_trigger*). Afin de faciliter l'extension du projet, chaque mini-jeu (*Maze, MemoryGame, FlechetteGame, MusicGame*) possède son propre modèle indépendant.
- **La Vue :** Elle prend en charge l'affichage graphique à l'écran. Chaque page implémente ses propres fonctions *init_, draw_* et *free_*. Pour optimiser les performances, les ressources graphiques sont chargées une unique fois à l'initialisation et réutilisées à chaque “frame”, évitant des rechargements coûteux.
- **Le Contrôleur :** Il intercepte les événements de la boucle principale (clavier, souris) et délègue les mises à jour logiques au modèle. La boucle principale, située dans *main.c,* synchronise ainsi la logique temporelle (*app_update*), la gestion des entrées et le rendu.

Choix technologiques et bibliothèques

Le développement en langage C a nécessité l'intégration de plusieurs bibliothèques externes afin de répondre aux exigences du cahier des charges tout en garantissant la portabilité du jeu sur Linux, macOS et Windows :

- **Affichage et son :** La bibliothèque SDL2 a été privilégiée pour sa maturité, sa documentation abondante et sa capacité à gérer efficacement les éléments 2D (images, formes, textes) ainsi que l'audio.
- **Persistance des données :** Pour sauvegarder localement l'historique des scores de manière propre et structurée, la solution d'une base de données légère SQLite3 a été préférée aux fichiers textes bruts. Elle permet de structurer les données dans un fichier local .db et de les manipuler facilement via des requêtes SQL (tris, sélections).
- **Structures de données :** Le besoin d'un système clé-valeur (dictionnaire) pour le positionnement sur la carte a conduit à l'adoption de la bibliothèque *uthash*, qui permet d'implémenter des tables de hachage directement sur des structures standards en C. En effet, le positionnement du joueur nécessite une recherche très fréquente du lieu à partir d’un identifiant.

Développement de la manette dédiée (Périphérique HID)

En parallèle du code du jeu, la seconde équipe a conçu un périphérique d'interaction sur mesure : une manette USB reconnue nativement comme un clavier (norme HID). Développé sous Linux via *STM32CubeMX* et *STM32CubeIDE*, le système repose sur une carte STM32 Nucleo F446RE et intègre deux composants principaux :

- **Le joystick :** Géré par le convertisseur analogique-numérique ADC1, il lit les valeurs de deux potentiomètres (axes X/Y). Ces valeurs sont comparées à des seuils de zone morte "*deadzone"* pour émuler le comportement des flèches directionnelles du clavier.
- **Le clavier matriciel (4 lignes x 3 colonnes) :** Son balayage régulier permet de détecter l'activation des touches. Les codes correspondants "*keycodes"* sont stockés dans une matrice d'état "*KEYMAP*".

La transmission des données vers l'ordinateur est assurée par l'API *USBD_HID_SendReport* à travers un port USB soudé au circuit. La mémorisation des états précédents permet d'éviter les envois redondants d'informations et assure une interaction fluide. La compréhension fine de cette architecture matérielle a été facilitée par la documentation technique mise à disposition par Mr. Bazin.

II. VALIDATION ET RESULTATS

La phase de validation a pour but de s'assurer que le jeu fonctionne correctement, sans bugs, et qu'il respecte les contraintes techniques exigées.

Tests unitaires

Dans une architecture MVC, il est essentiel de vérifier que la logique interne (le Modèle) est fiable avant d'ajouter l'affichage. Des tests unitaires ont été réalisés avec la bibliothèque CMocka. Ces tests couvrent en priorité la gestion du joueur (déplacement, maintien des jauges de stress et de fatigue dans leurs limites) ainsi que les mécaniques des mini-jeux (calcul des scores aux fléchettes, chronomètre du Memory, conditions de victoire et de défaite). L'analyse de la couverture de code, générée via gcov, indique des taux de couverture variant de 70 % à 100 % selon les modules concernés. Si la couverture n'est pas absolue sur l'ensemble du projet, la grande majorité des fonctions critiques du Modèle a été éprouvée. Ce résultat démontre que les cas d'usage principaux et les règles fondamentales du jeu ont été exécutés et validés avec succès lors des phases de test.

Validation fonctionnelle globale

L'interface (Vue) et les commandes (Contrôleur) ont ensuite été connectées au Modèle en respectant l'architecture MVC. Les tests fonctionnels ont consisté à jouer à l'application complète pour vérifier son comportement global. Le système gère correctement les différentes transitions : l'interaction

avec un bâtiment sur la carte charge le bon mini-jeu, et la fin de l'épreuve ramène le joueur sur la carte principale en conservant son score et ses statistiques.

Analyse de la mémoire avec Valgrind

Bien qu'une politique de gestion de mémoire ait été implémentée (utilisation systématique de free et des fonctions de destruction SDL à la fermeture de chaque page et mini-jeu), le rapport final de l'analyse de mémoire avec Valgrind relève la persistance de plusieurs warnings sur des fuites résiduelles (Defintely/Possibly Lost). Malgré nos efforts de débogage, la complexité du suivi des pointeurs entre les différentes couches de l'architecture MVC n'a pas permis de corriger l'intégralité de ces warnings.

Bilan et perspectives

Le projet répond aux exigences du cahier des charges : travail en équipe coordonné via Git, respect de l'architecture MVC, documentation générée avec Doxygen, tests unitaires et analyse de mémoire réalisés.

Plusieurs améliorations restent toutefois envisageables : pour rendre la simulation plus réaliste, le jeu pourrait intégrer de nouvelles caractéristiques, des mini-jeux supplémentaires ainsi qu'une difficulté progressive.

Rôle de l'Intelligence Artificielle

Tout au long du cycle de développement, l'intelligence artificielle a été intégrée comme un outil d'assistance et de support. Elle a grandement contribué à accélérer la prise en main de la bibliothèque SDL2 (notamment pour la partie Vue), créer des assets graphiques, à optimiser les phases de débogage et à concevoir les tests unitaires pour valider la robustesse du code.

CONCLUSION

Ce projet a dépassé le cadre de la simple programmation : les défis techniques, la gestion de la complexité et les contraintes de temps ont fait écho à la réalité de la période d'examens que nous cherchions à simuler.

L'association du langage C, de la carte STM32, d'une base de données SQL et de la bibliothèque SDL2 nous a imposé une rigueur de développement exigeante. Gérer manuellement la mémoire, structurer le code, orchestrer une boucle de jeu stable, synchroniser les événements et interfacer la sauvegarde de données nous ont permis d'acquérir une compréhension solide des fondations du développement logiciel.

Notre plus grande réussite Le résultat est un jeu fonctionnel et cohérent. Nous sommes particulièrement fiers d'avoir interconnecté les mécaniques de jeu (stress, fatigue, savoir) aux actions du joueur, créant ainsi un lien fort entre narration et système de jeu.

Cependant, la confrontation avec notre planning initial a révélé des écarts. Nous avons sous-estimé la complexité liée à la création des mini-jeux et à la communication entre les périphériques externes et le logiciel, ce qui a nécessité plus de temps que prévu.

Axes d'amélioration Pour aller plus loin, plusieurs pistes d'évolution sont envisageables :

- Enrichir le contenu avec de nouveaux mini-jeux et une difficulté progressive.
- Optimiser la fluidité des déplacements du personnage.
- Rendre l'interface graphique plus esthétique et raffinée.

INSA Rennes
20 avenue des Buttes de Coësmes
CS 70839
35708 Rennes cedex 7
Tél : + 33 (0)2 23 23 82 00
www.insa-rennes.fr

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES