Département Télécommunications

1 ère Année

2023/ 2024

Rapport de Projet de Programmation PG110 :

Pac-Man

Un Projet réalisé par : M. Khabouze Mehdi et M. Raji Oussama

Et encadré par : M. Swartvagher Philippe et Mme. Teboulbi Amina

Tâche 1 : Makefile

Nous avons créé un Makefile pour automatiser la compilation du projet. Il utilise gcc comme compilateur et définit les options de compilation et de liaison nécessaires pour les bibliothèques SDL2 et SDL2_image. La règle "all" compile le programme pacman à partir du fichier source principal "main.c", tandis que la règle "clean" supprime tous les fichiers générés lors de la compilation, y compris l'exécutable "pacman". En tapant simplement 'make' dans le terminal, le projet est compilé et l'exécutable est généré, et 'make clean' permet de nettoyer le projet en supprimant les fichiers temporaires, assurant ainsi une expérience de développement fluide et efficace.

Tâche 2 : Pac-Man et les murs du labyrinthe

Pour restreindre les déplacements de Pac-Man dans le labyrinthe, nous avons mis en place un mécanisme de gestion des collisions. Avant chaque déplacement de Pac-Man, nous vérifions si celui-ci peut se déplacer dans la direction souhaitée. Pour ce faire, nous avons implémenté une fonction can_move, qui prend en compte la position actuelle de Pac-Man ainsi que la direction dans laquelle il souhaite se déplacer. Cette fonction examine la case adjacente dans la direction donnée et vérifie si elle représente un chemin libre ou un mur. Si la case adjacente est un mur, le déplacement est interdit, sinon Pac-Man peut se déplacer dans cette direction.

Afin de faciliter le changement de direction, nous avons également sauvegardé la direction souhaitée lorsque l'utilisateur appuie sur une touche de direction. Puis, avant de mettre à jour la position de Pac-Man, nous vérifions si la direction souhaitée est

faisable. Si c'est le cas, la direction suivie par Pac-Man est mise à jour vers la direction souhaitée, sinon Pac-Man conserve sa direction actuelle. Ce mécanisme permet de retenir la nouvelle direction souhaitée et de l'appliquer quelques tuiles plus tard, lorsqu'il devient possible de se déplacer dans cette direction.

Voici une clarification des algorithmes utilisés sous forme d'une écriture algorithmique:

Fonction can_move(map, position, direction)

Entrées :

map : grille représentant le labyrinthe

position : position actuelle de Pac-Man

direction : direction dans laquelle Pac-Man souhaite se déplacer

Sortie : booléen indiquant si Pac-Man peut se déplacer dans la direction donnée

Variables locales : tile_size : taille d'une tuile dans la grille

Début

// Récupérer les coordonnées de Pac-Man dans la grille

pacman_x ← position.x / tile_size pacman_y ← position.y / tile_size

La division par tile_size est effectuée pour convertir les coordonnées de position de Pac-Man de pixels à des coordonnées de cases dans la grille du labyrinthe.

// Vérifier la possibilité de déplacement selon la direction

Selon direction faire

Cas HAUT :

Si map[pacman_y - 1][pacman_x] est un mur alors

Retourner faux

Fin Si

Cas BAS :

Si map[pacman_y + 1][pacman_x] est un mur alors

Retourner faux

Fin Si

De même pour les **cas DROITE / GAUCHE**

Fin Selon

Retourner vrai // Si aucune collision détectée, Pac-Man peut se déplacer

Fin

Difficultés rencontrées : Alignement de Pac-Man

Pour garantir que Pac-Man reste aligné au centre du chemin du labyrinthe, nous avons utilisé une approche simple mais efficace dans la fonction ‘can_move’ :

Avant de vérifier si Pac-Man peut se déplacer dans une certaine direction, on divise ses coordonnées par la taille d'une tuile (tile_size). Cela permet de transformer les coordonnées des pixels en coordonnées de tuiles, donc on obtient les indices des cases dans la grille du labyrinthe, où chaque case représente une tuile. ce qui facilite la détection de la présence de murs ou de chemins dans le labyrinthe. Cette transformation assure que Pac-Man reste toujours aligné avec le centre des cases du labyrinthe, ce qui résout le problème d'alignement.

Tâche 3 : Orientation de Pac-Man

Nous avons implémenté la fonctionnalité permettant à Pac-Man de toujours pointer dans la direction où il se déplace. Pour ce faire, nous avons chargé quatre textures différentes, chacune représentant Pac-Man dans une direction spécifique : vers le haut, le bas, la gauche et la droite. Ces textures ont été chargées à partir des fichiers image correspondants à l'aide des fonctions de la bibliothèque SDL2, plus précisément, avec la fonction **IMG_LoadTexture**(renderer, ''chemin_de_image'').

En changeant l'image de Pac-Man à chaque déplacement (UP, DOWN, RIGHT et LEFT) dans la boucle de jeu, nous avons assuré une synchronisation directe entre la direction de son déplacement et son orientation visuelle. Cela permet une expérience de jeu fluide et intuitive.

Tâche 4 : Ajout des fantômes

Pour réaliser cette tâche, nous avons utilisé plusieurs étapes et algorithmes :

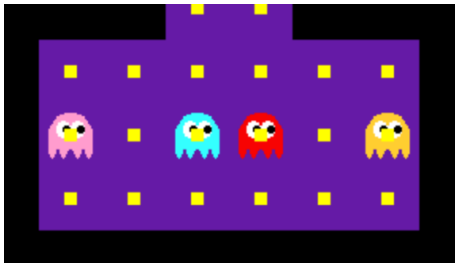
- **Chargement des images des fantômes :** Nous avons chargé les images des fantômes dans différentes directions à l'aide de la fonction **IMG_LoadTexture**.
- **Gestion des mouvements des fantômes :** Les fantômes se déplacent sur la grille du labyrinthe en choisissant aléatoirement leur prochaine direction parmi les choix possibles à chaque intersection et ce grâce à la fonction

choose_random_direction. Ensuite, nous avons utilisé la fonction **update_ghost_position** pour mettre à jour la position de chaque fantôme en fonction de sa direction actuelle et des collisions éventuelles avec les murs du labyrinthe. Voici l'algorithme de la fonction **update_ghost_position :**

Fonction update_ghost_position(ghost_position, previous_direction, map, pacman_position) :

Entrées :

- ghost_position : la position actuelle du fantôme
- previous_direction : la direction précédente du fantôme
- Map : la carte représentant le labyrinthe
- pacman_position : la position actuelle de Pac-Man



Variables : - tile_size : la taille d'une tuile dans le labyrinthe

Calculer la position du centre de Pac-Man : pacman_center_x = pacman_position.x + PACMAN_SIZE / 2

Choisir aléatoirement une direction pour le fantôme en excluant la direction précédente

random_direction = choisir_direction_aléatoire(previous_direction)

Mettre à jour la position du fantôme en fonction de la direction choisie :

Selon random_direction :

- Si UP :

Vérifier si le fantôme peut se déplacer vers le haut avec la fonction can_move (voir tâche 2)

Si oui, déplacer le fantôme vers le haut de tile_size et mettre à jour la direction précédente à UP

- Si DOWN :

Vérifier si le fantôme peut se déplacer vers le bas avec la fonction can_move (voir tâche 2)

Si oui, déplacer le fantôme vers le bas de tile_size et mettre à jour la direction précédente à DOWN

- Si LEFT :

Vérifier si le fantôme peut se déplacer vers la gauche avec la fonction can_move (voir tâche 2)

Si oui, déplacer le fantôme vers la gauche de tile_size et mettre à jour la direction précédente à LEFT

- Si RIGHT :

Vérifier si le fantôme peut se déplacer vers la droite avec la fonction can_move (voir tâche 2)

Si oui, déplacer le fantôme vers la droite de tile_size et mettre à jour la direction précédente à RIGHT

→ Voici l'algorithme de la fonction **choose_random_direction :**

Fonction choose_random_direction(previous_direction):

Entrée : previous_direction : la direction précédente du fantôme

Variables : random_direction : la direction aléatoire choisie pour le fantôme

Sortie : la direction aléatoire choisie pour le fantôme

//Obtenir un nombre aléatoire entre 0 et 3 inclusivement : random_direction = rand() % 4

Tant que la direction aléatoire est opposée à la direction précédente :

Si (random_direction == HAUT et previous_direction == BAS) OU

(random_direction == BAS et previous_direction == HAUT) OU

(random_direction == GAUCHE et previous_direction == DROITE) OU

(random_direction == DROITE et previous_direction == GAUCHE) alors

Choisir une autre direction aléatoire

random_direction = rand() % 4

Fin Si

Fin Tant que

Retourner la direction aléatoire choisie

```
Pac-gums eaten: 10
Pac-Man collided with a ghost!
```

- **Détection des collisions avec Pac-Man :** Nous avons inclus une vérification des collisions entre la position de Pac-Man et celle de chaque fantôme à chaque itération de la boucle principale du jeu. Cela se fait à l'aide de la fonction **check_pacman_ghost_collision.** Voici son algorithme :

Fonction check_pacman_ghost_collision(pacman_position, ghost_position):

Entrées :

- pacman_position : la position de Pac-Man (rectangle)
- ghost_position : la position du fantôme (rectangle)

Variables : collision : un indicateur de collision

//Vérifier si les rectangles se chevauchent :

Si (pacman_position.x < ghost_position.x + ghost_position.w) **ET**
(pacman_position.x + pacman_position.w > ghost_position.x) **ET**
(pacman_position.y < ghost_position.y + ghost_position.h) **ET**
(pacman_position.y + pacman_position.h > ghost_position.y) alors
collision = 1 **// Collision détectée**
Sinon
collision = 0 **// Pas de collision**
Fin Si
Retourner l'indicateur de collision

- **Pause du jeu en cas de collision :** Si une collision entre Pac-Man et l'un des fantômes est détectée, le jeu est mis en pause en activant le mode is_paused, ce qui gelera l'affichage et empêchera les déplacements de Pac-Man. Selon l'algorithme suivant :

Si check_pacman_ghost_collision(pacman_position, red_ghost_position) **OU**
check_pacman_ghost_collision(pacman_position, blue_ghost_position) **OU**
check_pacman_ghost_collision(pacman_position, orange_ghost_position) **OU**
check_pacman_ghost_collision(pacman_position, pink_ghost_position) ALORS
is_paused ← 1 **// Mettre le jeu en pause**
et les fantômes reviennent à leur base (m-à-j de leurs cooordonnées)
Afficher "Pac-Man collided with a ghost!" sur le terminal
Fin Si

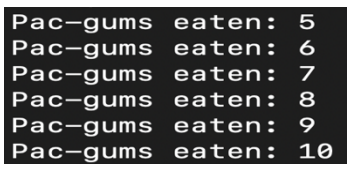
- **Reprise du jeu :** Nous avons permis au joueur de reprendre le jeu en appuyant sur la touche "r" après une collision à l'aide des fonctions event.key.keysym.sym et SDLK_r de la bibliothèque SDL2. Cela réactive le jeu en désactivant le mode pause (is_paused = 0).

Dans la boucle principale du jeu, nous avons utilisé les fonctions que nous avons précédemment définies pour mettre à jour la position des fantômes et vérifier s'il y avait une collision entre Pac-Man et les fantômes. À chaque itération de la boucle, nous avons appelé la fonction update_ghost_position pour mettre à jour la position de chaque fantôme en fonction de son mouvement aléatoire. Ensuite, nous avons vérifié s'il y avait une collision entre la position actuelle de Pac-Man et la position de chaque fantôme à l'aide de la fonction check_pacman_ghost_collision. Si une collision était détectée avec l'un des fantômes, nous avons mis en pause le jeu en définissant le drapeau is_paused sur 1, ce qui arrêtait le mouvement de Pac-Man et des fantômes. De plus, un message indiquant que Pac-Man avait heurté un fantôme était affiché sur la console à l'aide de printf.

→ Nous avons opté pour ces structures de données, algorithmes et logique car elles offrent une approche efficace et modulaire pour gérer les déplacements des personnages dans le jeu. Les fonctions telles que update_ghost_position et check_pacman_ghost_collision sont conçues pour être réutilisables et faciles à comprendre, notamment à la tache 7, ce qui simplifie le développement et la maintenance du code.

Tâche 5 : Pac-gommes

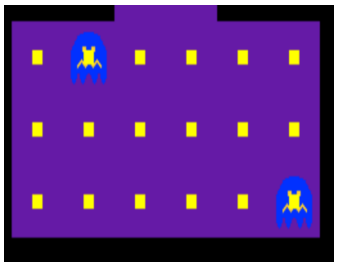
Nous avons ajouté les pac-gommes au sein du labyrinthe. Pour ce faire, nous avons parcouru la grille représentant le labyrinthe et dessiné un rectangle jaune pour chaque case de chemin avec les fonctions : SDL_SetRenderDrawColor et SDL_RenderFillRect. afin de symboliser une pac-gomme. Lorsque Pac-Man passe sur une pac-gomme, celle-ci disparaît et le nombre total de pac-gommes mangées est incrémenté. Cette implémentation a été réalisée en mettant à jour la grille du labyrinthe pour représenter une case vide (-1) à l'emplacement d'une pac-gomme mangée, tout en affichant le nombre de pac-gommes mangées sur le terminal.



Tâche 6 : Miamm les fantômes 😋 !

Dans le cadre de la tâche 6, nous avons introduit la mécanique permettant à Pac-Man de manger les fantômes une fois qu'un certain nombre de pac-gommes ont été collectées. Lorsque Pac-Man a mangé entre 100 et 150 pac-gommes inclusivement, les fantômes deviennent vulnérables pendant une durée limitée. Pour refléter cet état de vulnérabilité, nous avons modifié la texture des fantômes pour qu'ils apparaissent en bleu. Si Pac-Man entre en collision avec l'un de ces fantômes vulnérables, le fantôme est "mangé", sa position est réinitialisée au centre du labyrinthe et sa texture d'origine réapparait après un bref délai.

Moment de la collision



Position réinitialisée

Tâche 7 : I.A des fantômes

Nous avons développé un algorithme pour diriger les fantômes vers Pac-Man. Cet algorithme repose sur la sélection de la direction qui permet aux fantômes d'atteindre Pac-Man le plus rapidement possible. Pour ce faire, nous avons défini trois fonctions distinctes : update_red_ghost_position, update_blue_ghost_position, et update_ghost_position, chacune responsable du déplacement d'un type de fantôme. L'algorithme utilisé commence par calculer la position actuelle de Pac-Man et celle du fantôme concerné. Ensuite, il sélectionne une direction pour le fantôme en fonction de sa position par rapport à celle de Pac-Man. Par exemple, si Pac-Man se trouve à gauche du fantôme, celui-ci choisira une direction vers la gauche pour se rapprocher de Pac-Man. Nous avons expérimenté ces deux métriques (Manhattan et Euclidienne) pour déterminer laquelle offre les meilleurs résultats en termes de mouvement stratégique des fantômes. En ajustant ces métriques, nous avons pu influencer le comportement des fantômes pour les rendre plus compétitifs et stimuler l'interaction avec Pac-Man.

Voici l'écriture algorithmique de la fonction update_ghost_position :

```
Fonction update_ghost_position(ghost_position, previous_direction, map, pacman_position):
  taille_case = 40
  Centre_de_Pacman_X = position_de_Pacman.x + TAILLE_PACMAN / 2
    direction_aléatoire = choisir_direction_aléatoire(previous_direction) ou bien choisir_direction_proche
      Selon direction_aléatoire faire
```

Cas HAUT : Si peut_se_déplacer(map, ghost_position, HAUT) alors ghost_position.y -= taille_case

previous_direction = HAUT **Fin Si**

De même pour les **Cas BAS / DROITE / GAUCHE**

Fin Selon

Fin Fonction

Voici l'écriture algorithmique de la fonction choisir_direction_proche :

Fonction choisir_direction_plus_proche_de_pacman(position_du_fantôme, position_de_pacman, carte) :

min_distance ← INFINI ;meilleure_direction ← HAUT

Pour chaque direction dir allant de HAUT à DROITE :

prochain_x, prochain_y ← position_du_fantôme + déplacement_de(dir)

Si peut_se_déplacer(carte, position_du_fantôme, dir) alors distance ← distance_de_Manhattan(prochain_x, prochain_y, centre_de_pacman_x, centre_de_pacman_y)

Si distance < min_distance alors

min_distance ← distance et meilleure_direction ← dir

Fin Si

Fin Si

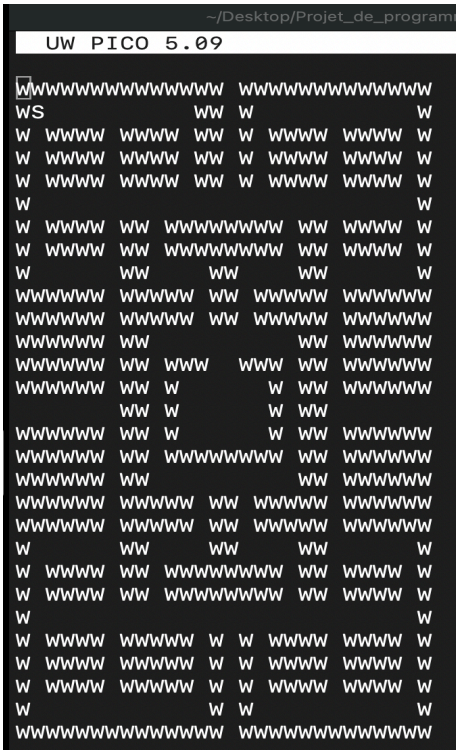
Fin Pour

Retourner meilleure_direction

Fin Fonction

Tâche 8 : charger le labyrinthe depuis un fichier

Nous avons implémenté une fonction nommée **load_maze**. Cette fonction prend en entrée le nom du fichier contenant la représentation du labyrinthe et remplit une matrice avec les types de tuiles correspondants. Chaque caractère dans le fichier est associé à un type de tuile dans le labyrinthe, par exemple, 'W' pour les murs, 'S' pour la position de départ de Pac-Man, 'G' pour la position initiale des fantômes et ' ' pour les chemins. Nous avons adopté une approche de lecture de fichier ligne par ligne, en analysant chaque caractère pour déterminer le type de tuile correspondant dans la matrice du labyrinthe. Cette méthode permet une flexibilité dans la conception des labyrinthes, car il est possible de les éditer simplement en modifiant le fichier correspondant sans avoir à modifier le code source. Une fois le labyrinthe chargé, il peut être utilisé pour initialiser le jeu, permettant ainsi une personnalisation aisée de l'environnement de jeu.



Exemple d'une map

Tâche 9 : Téléportation !!!

La tâche 9 a été une étape cruciale dans l'amélioration de notre jeu, permettant à Pac-Man et aux fantômes de se déplacer d'un côté à l'autre du labyrinthe à travers des ouvertures symétriques opposées. Cette fonctionnalité a ajouté une dimension stratégique au jeu, offrant aux joueurs la possibilité de traverser rapidement le labyrinthe pour échapper aux fantômes ou pour piéger ces derniers. L'implémentation de cette fonctionnalité s'est faite grâce à la création de points de téléportation positionnés aux extrémités opposées des ouvertures dans le labyrinthe. Lorsque Pac-Man ou un fantôme atteint l'un de ces points, ils sont instantanément téléportés vers l'ouverture opposée, leur permettant de se déplacer rapidement d'un côté à l'autre. Cette mécanique de jeu ajoute de la variété et de l'excitation aux déplacements dans le labyrinthe, renforçant ainsi l'expérience de jeu pour les joueurs. Voici l'algorithme de la téléportation :**Fonction teleport_pacman(position_de_pacman, ouverture):**

Selon ouverture faire

Cas HAUT : position_de_pacman.y ← 1043 Position y du point de téléportation au bas

Cas BAS : position_de_pacman.y ← 45 Position y du point de téléportation en haut

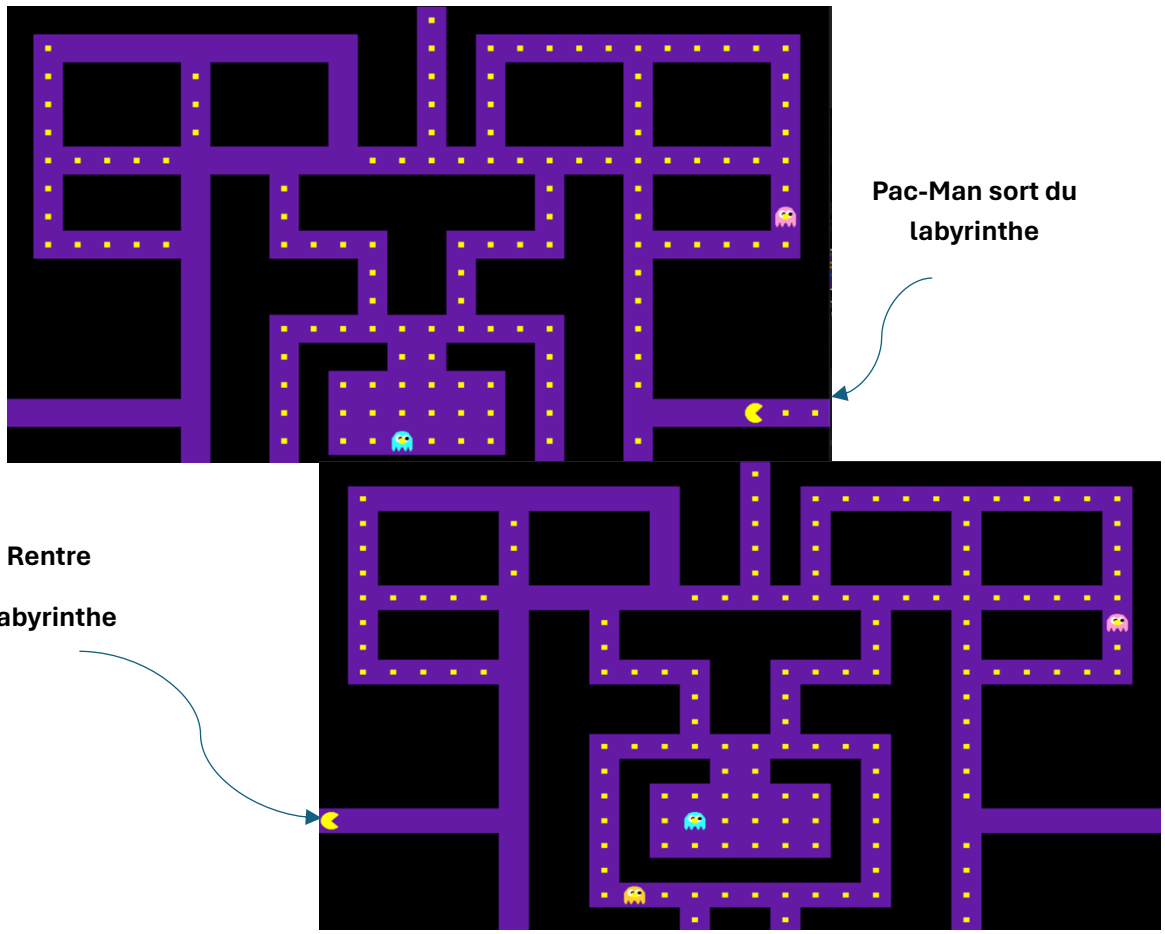
Cas GAUCHE : position_de_pacman.x ← 1043 Position x du point de téléportation à droite

Cas DROITE : position_de_pacman.x ← 2 Position x du point de téléportation à gauche du labyrinthe

Défaut : Afficher "Ouverture invalide spécifiée"

Fin Selon

Fin



→ Ensuite dans la boucle du jeu, nous avons utilisé la logique suivante :

```
//Déclaration des coordonnées pour les portes de téléportation
  droite.x ← 27      gauche.x ← 0      haut.x ← 14      bas.x ← 14
  droite.y ← 14      gauche.y ← 14     haut.y ← 0       bas.y ← 28
Si pacman_tile_x == droite.x et pacman_tile_y == droite.y alors
  Téléporter Pac-Man à droite
Sinon si pacman_tile_x == gauche.x et pacman_tile_y == gauche.y alors
  Téléporter Pac-Man à gauche
De même pour les Cas HAUT / BAS
Fin Si
```

Conclusion :

Mehdi : Après avoir terminé ce projet, je ressens un mélange de satisfaction et de fierté. J'ai vraiment appris énormément de choses tout au long de ce processus. En plus des compétences techniques liées à la programmation en C et à l'utilisation des bibliothèques SDL2 et SDL2_image (qui ont été très pénibles à exploiter sur MacOS), j'ai également acquis une précieuse leçon de patience. Rencontrer des erreurs et des obstacles m'a permis de développer ma capacité à résoudre des problèmes de manière efficace et créative.

Ce projet m'a également offert une perspective plus profonde sur le processus de développement logiciel. J'ai pu voir comment les différentes parties d'un programme interagissent les unes avec les autres et comment la planification et l'organisation sont essentielles pour mener à bien un projet complexe.

De plus, ce projet m'a donné l'opportunité d'apprendre à travailler en équipe. Collaborer avec Oussama a été une expérience enrichissante, où nous avons partagé nos connaissances et résolu des problèmes ensemble. De plus, l'utilisation de Git comme outil de gestion de version nous a permis de coordonner efficacement notre travail et de suivre les différentes versions du code.

Je tiens également à exprimer ma gratitude envers nos professeurs encadrants. Leur soutien et leurs conseils précieux ont été d'une aide inestimable tout au long de ce projet.

En rétrospective, bien que j'aie fait beaucoup d'erreurs et rencontré des défis tout au long du chemin, chaque obstacle surmonté m'a permis de progresser et de devenir un meilleur développeur. Ce projet a été une expérience enrichissante qui m'a donné la confiance nécessaire pour aborder des projets futurs avec détermination, curiosité et une solide base de collaboration et de gestion de version.